Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Рыжкова Елизавета Евгеньевна

Выпускная квалификационная работа бакалавра

Разработка автоматизированной школьной системы оценки результатов качества образования на платформе Microsoft Azure

Направление 010300

Фундаментальные информатика и информационные технологии

Научный руководитель:
PhD,
доцент
Корхов В. В.

Санкт-Петербург
2016

Оглавление

Введение

На сегодняшний день важность образования неоспорима. Однако не всегда результаты образовательного процесса соответствуют нормативным требованиям, личным и социальным ожиданиям. Степень удовлетворительности этих результатов принято называть качеством образования [1].

Оценка качества образования включает в себя мониторинг как результатов обучения, так и самого образовательного процесса. Данный этап является необходимым для выявления существующих проблем в процессе обучения, соответствующих причин и их последующего устранения. Эти действия являются необходимыми для повышения качества образования.

В настоящее время в нашей стране уделяется немалое внимание оценке качества образования в образовательных учреждениях разного уровня, в том числе в школах. Данный факт отображается в ряде нормативных и распорядительных документов. Закон об образовании [2] устанавливает обязанности школы на проведение внутренней системы оценки качества образования. Федеральная целевая программа развития образования на 2016-2020 годы предусматривает развитие и совершенствование общероссийской системы оценки качества образования (СОКО)[3] на основе региональных и муниципальных.

Также в Законе об Образовании определены место и роль мониторинга в системе образования [4]. Показатели мониторинга системы установлены образования для общего, профессионального и дополнительного образования, а также профессионального обучения приказом МОиН РФ № 14 от 15.01.2014 «Об утверждении показателей мониторинга системы образования» [5]. Правила осуществления мониторинга системы образования и перечень обязательной информации о системе образования, подлежащей мониторингу, утверждены

ранее постановлением Правительства Российской Федерации от 05.08.2013 № 662 «Об осуществлении мониторинга системы образования» [6], вступившим в силу с 01.09.2013.

Проведение внутреннего мониторинга школой подразумевает следующие этапы: сбор необходимых данных, подсчет показателей, необходимых для мониторинга, составление отчетов, содержащих результаты мониторинга, передача их в органы исполнительной власти, анализ экспертом (администрацией школы) отчетов на основе своего опыта и знаний с целью получения вывода о существующих проблемах и возможных причинах в обучении, принятие административных решений руководством школы.

Мониторинг системы образования осуществляется на основе многих параметров, в т. ч. результаты образовательного процесса, условия образования, материально-техническое и финансовое обеспечение и другие.

Так как целью любого образовательного процесса является приобретение знаний учащимися, то можно считать наиболее существенной частью мониторинга оценку качества результатов образования (конечных оценок), на которой и предлагается сконцентрироваться в рамках данной работы. Система оценки качества результатов образования, как и СОКО, подразумевает определение и вычисление показателей и значений, которые будут свидетельствовать о развитии образовательного учреждения, наличии либо отсутствии проблем в образовательном процессе.

Распространённое решение, все ещё использующееся во многих школах – перенос данных с бумажных носителей вручную в excel и произведение необходимых расчетов в данном редакторе. Такой подход имеет очевидный недостаток – затраты времени.

Процесс оценки качества образования может быть автоматизирован. Система с данной функциональностью должна удовлетворять следующим критериям:

1. Наличие электронного хранилища данных, которое включает в себя всю необходимую информацию для подсчета показателей, характеризующих результаты образовательного процесса.

2. Данные в хранилище должны пополнятся естественно и регулярно, – именно выполнение данного пункта необходимо для избавления от проблемы временных затрат. Для подобной реализации необходим пользовательский интерфейс, предоставляющий возможность пополнения хранилища новыми данными, их обновления, удаления, вывода, запроса и отображение показателей СОКО.

3. Функциональность подсчета названных показателей.

Автоматизация процесса оценки качества образования позволит сократить временные затраты на создание отчетов о мониторинге, позволит сконцентрироваться администрации школ на решении других задач.

Однако при наличии такой системы ответственные лица все еще вынуждены тратить много времени на проведение анализа полученных отчетов: выявление проблем, поиск взаимосвязей между найденными проблемами и образовательным процессом, выявление причин существующих проблем. Также есть вероятность, что человеческий фактор станет причиной неполного анализа, что воспрепятствует повышению качества образовательных услуг.

Таким образом, необходимо не только программное составление отчетов для школьной СОКО (ШСОКО), но и автоматизированная система для поиска проблем в образовательном процессе и соответствующих им причин.

Постановка задачи

Целью данной работы является повышение эффективности процесса оценки качества образования и поиска причин неудовлетворительных показателей качества образования в школе.

Существует ряд программных средств, которые подробнее будут рассмотрены в главе 1, позволяющих получать показатели качества результатов образования. Однако данные средства не представляют сколько-либо существенных возможностей для анализа полученных показателей и поиска причин неудовлетворительных показателей.

Для достижения обозначенной цели необходимо выполнить следующие задачи:

- Изучить предметную область и разработать систему правил для нахождения проблем в образовательном процессе и их предполагаемых причин;
- Разработать прототип программного решения для осуществления поставленной цели на основе разработанной системы правил.

Главная задача данного программного обеспечения – автоматизировать процесс поиска причин проблем в образовательных компонентах школы. Так как проблемами в образовательном процессе считаются неудовлетворительные показатели качества результатов образования, программное обеспечение должно обладать функционалом их подсчета и вывода.

Для поиска проблем обучения и их причин необходимо определенное хранилище данных, которое должно содержать актуальные данные. Так как главная цель – автоматизация, то не должно быть временных задержек, затрачиваемых на перенос данных из других источников в хранилище для анализа. Поэтому хранилище данных должно пополняться естественно и регулярно, с учетом данной необходимости наилучшем решением будет

реализация функций электронного журнала. Функциональность электронных журналов рассмотрена в главе 1.

Хотя перед разрабатываемым прототипом не стоит задача конкуренции с существующими решениями электронных журналов, и данные функции будут реализовываться лишь по мере необходимости для решения поставленных задач, при проектировании системы должны быть учтены возможности расширения данных функций.

Обзор литературы

Информация из [1-6], [14] использовалась для изучения официальных данных об оценке качества образования в Российской Федерации. Ссылки содержат законы и приказы, регламентирующие деятельность образовательных учреждений по оценке качества образования, а также общую справочную информацию о принципах оценки качества образования.

Источники [7-10] содержат найденные существующие коммерческие решения, которые позволяют решать часть задач, поставленных в данной работе.

[11-13] использовались при поиске похожих решений на описанное в данной работе. Эта литература включает в себя статьи, в которых освещается применение экспертных систем, систем правил в образовании. Найденная информация помогла в понимании общих аспектов при разработке решений систем правил в области образования.

Источники [16], [33-39] содержат информацию, используемую для изучения возможностей Microsoft Azure. [16] содержит общую необходимую информацию. В [33] рассматриваются планы размещения в службе приложений Azure. [34-36] поясняют работу масштабирования. [37] содержит информацию о базах данных Azure SQL. В [38-39] даются инструкции по резервному копированию и восстановлению в Azure.

[15], [17-32], [40-42] – это ссылки на официальную документацию программных продуктов, которые были рассмотрены или применены в рамках данной работы.

Глава 1. Существующие решения

В связи с актуальностью проблемы существует ряд продуктов, автоматизирующих часть мониторинга образовательного процесса. Данные решения можно разделить на группы согласно их функциональности: электронные журналы, электронные журналы с функциональностью статистических отчетов, системы для автоматического анализа образовательного процесса.

1.1. Электронные журналы

Электронный журнал – это некоторое программное обеспечение, включающее электронное хранилище данных и интерфейс для взаимодействия с пользователями, позволяющее заменить стандартные бумажные школьные журналы на электронные. Подобные решения, как правило, обладают следующими возможностями: предоставление расписания уроков, выставление оценок за уроки и учебные периоды, управление домашними заданиями, дневник ученика, контроль успеваемости учеников со стороны их родителей, оповещения о событиях, хранение и предоставление персональной информации об учениках и учителях, администрации школы и прочее.

На сегодняшний день можно отметить существенное развитие электронных журналов. Например, подобной функциональностью обладают «ЭлЖур» [7], «SmileS.Школьная карта» [8], «Дневник.ру» [9].

Большинство школ РФ используют подобные приложения в той или иной мере.

1.2. Электронные журналы с предоставлением отчетов по качеству образования

Данные приложения представляют собой решения в виде электронных журналов, описанных в предыдущем пункте. Но, кроме того, эти программы обладают функциональностью получения показателей качества образования.

Названные функции также реализованы в некоторых коммерческих продуктах. В том числе «ЭлЖур», «Дневник.ру», «ИРТех» [10].

«ИРТех» предоставляет следующие возможности: просмотр результатов тестирований, оценок за учебный период, просмотр показателей качества знаний, успеваемости, среднего балла, общий оценочный балл, количество учеников с одной тройкой и т. д. Просмотр названных значений производится по классам, по предметам, по учителям, по классным руководителям. Также представлен прогноз оценок, рейтинг учителей на основе успеваемости.

Почти аналогичен описанному функционал электронного журнала «ЭлЖур». А «Дневник.ру» имеет несколько меньше показателей мониторинга для расчета.

Несмотря на достаточно широкий спектр показателей качества образования, описанные решения не предлагают никакого инструментария для анализа полученных показателей и поиска причин неудовлетворительных показателей.

1.3. Решения для выявления причин неудовлетворительных показателей качества образования

Как уже было сказано, администрация школы (т. е. эксперт) в процессе оценки качества образования подсчитывает вручную показатели, характеризующие наличие проблем в обучении, либо использует для этого подходящее программное обеспечение. Несоответствие показателей качества образования нормативным требованиям считается проблемой. Далее эксперт должен проанализировать полученные показатели и сопоставить их с имеющейся информацией об образовательном процессе. На основе своих знаний и опыта администрация школы делает выводы о причинах, которые

могли вызвать найденные проблемы. Данный процесс может быть длительным, трудоемким и сложно выполнимым вручную. Велика вероятность ошибки. Длительная обработка может являться причиной получения неактуальной информации или потерей информации в ходе её обработки.

Однако на сегодняшний день нет программных систем с функциональностью, позволившей бы автоматизировать процесс поиска причин неудовлетворительных показателей (далее по тексту работы неудовлетворительные показатели также будут назваться проблемами).

Одним из возможных решений данной задачи может быть экспертная система. Экспертная система – некоторая программная система, функциональность которой схожа с работой эксперта, и цель которой – полностью или частично заменить эксперта.

На основе опыта администрации школы может быть построена экспертная система, которая позволит автоматически находить возможные причины неудовлетворительного качества образования.

1.4. Другие применения экспертных систем в образовании

Существует множество теоретических подходов применения экспертных систем в образовании. В ряде публикаций [11] предлагаются различные подходы для решения следующих задач: диагностика по уровням обученности и профессиональным предпочтениям обучающихся, формирование индивидуальной траектории обучения в ходе самостоятельной работы учащегося за компьютером, создание индивидуальных учебных планов, диагностика знаний по психологии, анализ качества знаний обучающихся на основе структурированности знаний, обучение, оценка знаний обучающегося относительно знаний преподавателя, выбор университета для поступления, проведение SWOT-анализа вуза [12] и другие.

Кроме того, есть некоторые разработки экспертных систем для применения в области образования: диагностика качества обучения в вузе, оценка качества педагогической подготовки будущего учителя, адаптивное дистанционное обучение, оценка качества образовательной деятельности вуза с помощью экспертного оценивания, средство для изучения языка пролог для учащихся 9 класса, экспертная система оценки рисков IT-образования [13].

Часто в подходах реализации экспертных систем предлагается использовать методы нечеткой логики: вычисление комплексной успеваемости студентов (посещение лекций, выполнение контрольных, домашних заданий, работа на семинаре), профессиональная ориентация для студентов, мониторинг образовательного процесса вуза, где за определение качества образования берутся «прочность», «глубина» и «осознанность» знаний учащихся, которые определяются на основе нечеткой логики, «KnowledgeCT» – оценка знаний с помощью системы тестирования, подбирающая вопросы по уровню сложности, составление индивидуальной траектории обучения в зависимости от принадлежности обучающегося к определенной группе, дистанционное обучение.

Говоря в общем о применении экспертных систем в образовании, нельзя не обратить внимание на то, что большинство изучаемых и реализуемых экспертных систем в сфере образования либо нацелены на решения задач для вузов, либо их целью является построение индивидуальных рекомендаций для обучения, либо они реализуют какие-либо функции дистанционного обучения и тестирования знаний. Эти системы не подразумевают функциональность поиска проблем и их причин в образовательном процессе в школе или решения схожих задач.

Глава 2. Система правил для поиска причин неудовлетворительного качества образования

Примечание: описываемая система правил получена с помощью самостоятельного изучения предметной области и консультирования с администрацией одной из школ.

2.1. Потенциальные причины проблем

Список причин, которые можно выявить с помощью описываемой системы правил заранее определен:

- Низкий уровень компетентности педагогов;
- Неопытность учителей;
- Отсутствие мотивации;
- Устаревшие знания;
- Пенсионный возраст;
- Чрезмерная загруженность учителя;
- Недостаточный контроль со стороны классного руководителя;
- Завышение оценок;
- Занижение оценок / Завышенные требования к ученикам;
- Не оптимальный учебный план;
- Недостаток учебных часов;
- Невыполнение плана учебной программы;
- Низкая посещаемость учеников;
- Не проведение уроков;

- Отсутствие дополнительных занятий;
- Отсутствие внеурочной деятельности;
- Недостаток материально-технического обеспечения школы;
- Снижение общих показателей из-за отчисления успешных учеников или зачисления слабых учеников;
- Недостаток навыков решения определенных типов заданий у учеников;
- Отсутствие мотивации у учащихся;

Цель применения системы правил – найти одну или несколько причин из перечисленных.

2.2. Поиск проблем

Показатели, характеризующие качество результатов образования:

Качество знаний – отношение количества четверок и пятерок к общему количеству учеников.

Успеваемость – отношение количества троек, четверок и пятерок к общему количеству учеников.

Для данных параметров устанавливаются (местным самоуправлением, уставом школы) некоторые критические значения. Если показатели качества знаний или успеваемости (КЗУ) меньше установленных критических значений, то такое КЗУ считается ***проблемой*.**

На этапе поиска проблем нужно проверять КЗУ по заданному периоду по следующим группам (рис. 1):

- По всей школе;

- По каждому учителю;
- По каждому предмету;
- По учителю и предмету;
- По классному руководителю.

Рис. 1. Получение начальных показателей

При нахождении неудовлетворительного КЗУ на этом этапе считаем, что найдены ***явные проблемы.***

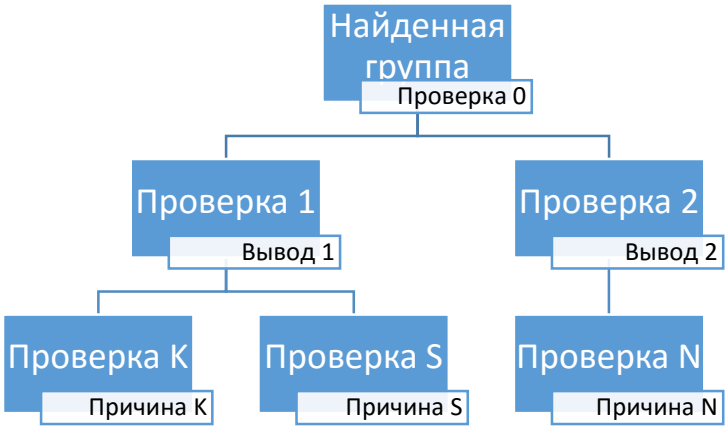
Если значения выше критических, то считаем, что явных проблем в школе нет и переходим к выводам на основе результатов тестирования.

2.3. Анализ явных проблем

Могут быть выявлены следующие проблемы:

- КЗУ меньше граничного по школе;
- КЗУ меньше граничного по одному или нескольким учителям;
- КЗУ меньше граничного по одному или нескольким предметам;
- КЗУ меньше граничного по учителю и предмету, который он ведет, но по предмету и учителю отдельно КЗУ удовлетворительное (по нескольким учителям и предметам);
- КЗУ меньше граничного по классному руководителю.

При выявлении перечисленных проблем проводиться анализ каждой из них отдельно на основе последовательных проверок фактов вида «ЕСЛИ – ТО» (рис. 2).



Процесс использования системы правил также подразумевает установку некоторых стандартных значений: удовлетворительный процент посещаемости и другие.

Рис. 2. Последовательная проверка фактов

Пусть ***КЗУ меньше заданного для учителя и предмета, но по предмету и учителю отдельно — выше критического значения.***

Для начала проверяется качество знаний и успеваемость рассматриваемой группы учеников по остальным предметам, если КЗУ ниже критического, то считаем, что у данных учащихся *нет мотивации к обучению*, дальнейший анализ не проводим.

Далее проверяется, сколько занятий преподаватель провел от общего числа уроков. Если данный преподаватель вел уроки совместно с другим учителем, то нужно сделать выводы на основе результатов тестирований. В случае, если КЗУ результатов тестов по темам, изучаемым с разными учителями не различаются более чем на 20% (см. пункт 2.4), то вопрос о качестве преподавания тем или иным учителем рассматривать не нужно.

Далее проверяется выполнение учебно-тематического плана. Каждой теме предмета сопоставляется необходимое для проведения число уроков. Если какое-либо количество уроков не было проведено, то делается вывод *о*

невыполнении образовательной программы. Если для каждой темы предмета было проведено регламентированное количество уроков, то образовательная программа выполняется.

Следующий шаг – просмотр посещаемости учеников с плохими отметками. Пользователь устанавливает удовлетворительное значение посещаемости, либо берется среднее по школе. Если уроки по предмету проводили несколько учителей, то имеет смысл смотреть посещаемость только у тех преподавателей, которые провели не менее 30% уроков от общего количества уроков курса. Если несколько учителей провели более 30% уроков, то смотрим посещаемость по каждому из учителей.

Если посещаемость учеников по данному курсу (предмету и учителю), меньше установленного значения, то нужно посмотреть посещаемость этой же группы учеников у других учителей и по другим предметам. Если посещаемость других курсов также меньше установленного значения, то можно делать вывод о *низкой посещаемости учеников и низкой мотивации*. Если посещаемость других курсов удовлетворительная, то делаем вывод об *отсутствии мотивации у педагога проведения занятий по данному предмету*, либо *низкой организацией и контролем учебного процесса со стороны администрации*. В случае с несколькими учителями, делаем эти выводы для учителя с соответствующими показателями посещаемости.

Нужно оценить нагрузку преподавателя, если количество рабочих часов больше установленных нормами для учителей в школе, то делаем вывод, что *у преподавателя превышена нагрузка* и это сказывается на качестве его работы. То же для всех преподавателей, которые вели этот предмет.

Смотрим результаты независимых тестирований, если за них оценки ниже или выше (на 2 балла у 25 процентов учеников и на 1 балл у 50 процентов учеников), то имеет место быть необъективное *занижение оценок или завышенные требования к ученикам*.

Еще одна потенциальная причина — *недостаточное количество часов по учебному плану*. Тогда рассматриваются ученики, у которых хорошие или отличные оценки по данному предмету у рассматриваемого учителя. Если в среднем у них в расписании больше часов по предмету, в том числе дополнительных и платных занятий, то наблюдается рассматриваемая причина. То же, если сократилось число занятий, и были удовлетворительные КЗУ.

Иначе делаем вывод о *некомпетентности учителей (если не менее 30% уроков проведено) в данном предмете*.

Схожий анализ производится при ситуациях: *КЗУ меньше по определенному учителю, КЗУ меньше по определенному предмету*, *КЗУ неудовлетворительное за весь период обучения.*

Более подробно разбор проблем не предоставляется, разбор проблем КЗУ меньше критического по учителям или предмету схож с ситуацией неудовлетворительных показателей по учителю и предмету одновременно, читателю предлагается ознакомиться со схемами сопоставления проблем и причин в приложении 1.

2.4. Вывод на основе результатов тестирований

При отсутствии ***явных*** проблем в обучении, выводы на основе результатов тестирований (рис. 3) могут стать основным инструментом в поиске решений для повышения качества образования. Основное преимущество – возможность поиска отдельных тем предметов, определенных проблемных типов заданий, с которыми у учеников возникли проблемы, и сопоставление этих тем с проведенными уроками, учителями, их проводившими и т. д. Без просмотра результатов тестирований не обойтись и при анализе явных проблем.

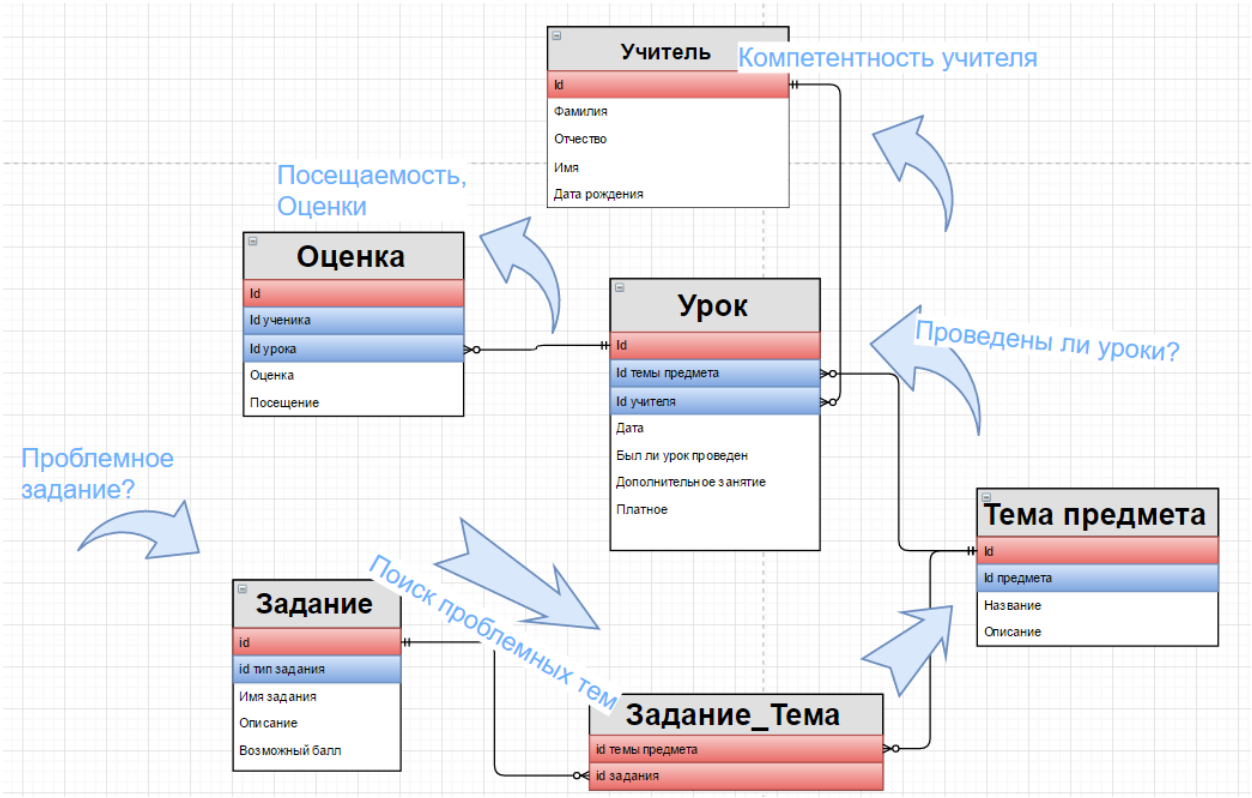


Рис. 3. Пример проверки информации.

Анализ схож с описанной в предыдущем пункте ситуацией «КЗУ меньше заданного для учителя и предмета, но по предмету и учителю отдельно — в пределах нормы», только в данном случае проводится по каждой теме и типу задания тестирования, КЗУ которых критическое.

2.5. Оценка компетентности преподавателя

На данную оценку влияют следующие факторы:

- Степень успешности обучения учеников данного преподавателя.
- Наличие аттестации — срок действия аттестации 5 лет. Исключением является молодой преподаватель после вуза, — он может работать без аттестации 3 года.
- Курсы повышения квалификации. В течение 5 лет учитель должен пройти курсы в размере не менее установленного количества часов [14].

- Наличие наград и званий. Их список ограничен, каждой награде или званию сопоставляется определенная степень значимости. Участие в конкурсных процедурах, в экспериментальной, инновационной, исследовательской, проектной деятельности.
- Является ли учитель одновременно классным руководителем. Успешность обучения учеников его класса.

Глава 3. Реализация

3.1. Архитектура и использование облачных технологий

Для разработки системы было выбрано веб-приложение по причине удобного доступа с разных платформ и отсутствия необходимости в процессе установки программ, что позволит максимально упростить работу сотрудникам школы.

Размещение системы автоматизированного мониторинга на школьном сервере потребовало бы определенных затрат на оборудование и квалифицированных специалистов для поддержания работы сервера. Во избежание данных проблем для разработки системы мониторинга процесса образования используются облачные технологии, в частности Microsoft Azure [15]. Такой подход гарантирует, что при поломке оборудования, оно будет заменено или починено без привлечения образовательного учреждения, и работа приложения не будет остановлена, либо период остановки будет незначительный. Также такое решение позволяет сократить сроки разработки.

Облачные вычисления – модель предоставления вычислительных ресурсов, при которой необходимые ресурсы интегрируются и предоставляются через Интернет как услуга.

Microsoft Azure предоставляет несколько сред выполнения приложений (рис. 4):

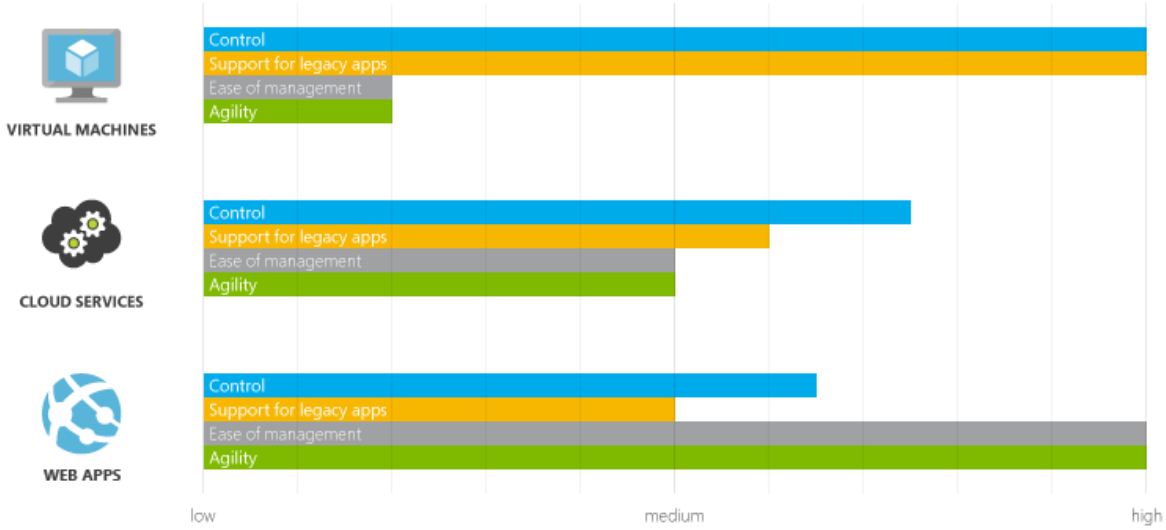


Рис. 4. Среды выполнения приложений Azure.

- Служба приложений – облачный сервис, который позволяет разработчикам создавать мобильные приложения, приложения API, приложения логики, а также веб-приложения, что является подходящим для разрабатываемого приложения;

- Облачные службы – предоставляют больший контроль над операционной системой;

- Виртуальные машины – виртуальные машины с полным контролем операционных систем (Windows или Linux) [16].

Функциональность облачных служб и виртуальных машин для разработки данного приложения не является необходимой и требует дополнительных затрат, поэтому был сделан выбор в пользу службы приложений.

Azure поддерживает веб-приложения, разрабатывающиеся на .NET [17], Node.js [18], PHP [19], Python [20] и Java [21]. Для описываемой программы был выбран .NET, в частности, язык программирования C# [22] и платформа ASP.NET MVC [23]. Подробней архитектура приложения описана в пункте 3.3.

Для просмотра ресурсов и управления ими Microsoft предоставляет специальный веб-интерфейс (рис. 5) – портал Azure [24].

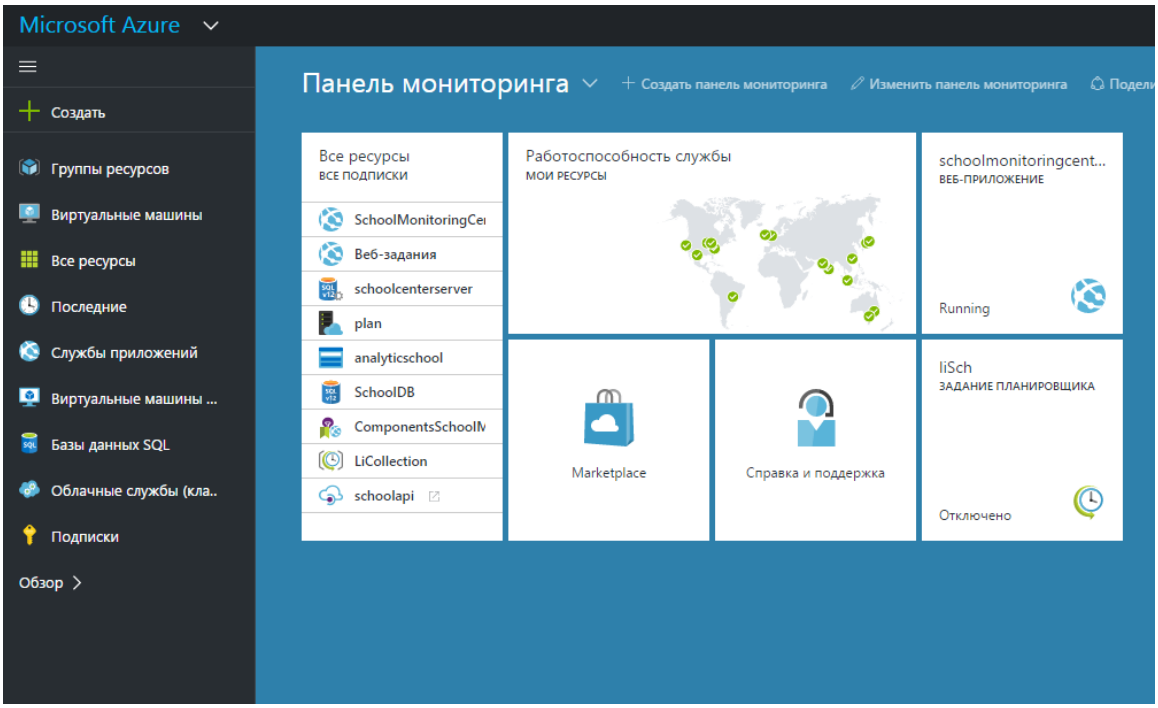


Рис. 5. Портал Azure

Служба приложений предлагает различные способы развертывания. Для данного сервиса используется функция «непрерывное развертывание». Оно представляет собой интеграцию с облачной службой управления версиями. Поддерживаются BitBucket [25], GitHub [26] и Visual Studio Team Services [27] и другими. В качестве службы управления версиями был выбран BitBucket. Для репозиториев Git [28] и Mercurial [29] не требуется дополнительная настройка. При публикации обновлений в одной из этих служб Azure автоматически извлекает их.

3.1.1. Хранилище данных

Для построения системы автоматизированного мониторинга качества образования в качестве хранилища данных наиболее подходящим вариантом является реляционная база данных по причине того, что данные имеют реляционную природу с большим количеством связей, а также структура хранимых данных заранее известна.

Azure предоставляет реляционную базу данных как услугу – база данных SQL Azure [30].

Работа с базой данных Azure SQL практически аналогична работе с локальной базой данных MS SQL [31]. Она поддерживает все необходимые функции СУБД, такие как транзакции, обеспечение целостности данных, одновременный доступ к данным нескольких пользователей и так далее. Поддерживается T-SQL. Доступ к базе данных может осуществляться с теми же технологиями, что и в случае с SQL Server. Также можно работать с этой базой данных в SQL Server Management Studio [32].

Так как Azure SQL – услуга PaaS («Платформа как услуга»), то данная служба берет на себя некоторые задачи администрирования, такие как обновление программного обеспечения, управление инфраструктурой оборудования.

3.1.2. Производительность и масштабируемость

Масштабирование веб-приложения

При настройке веб-приложения важным является параметр «всегда включено», который можно включить или отключить. При включении означает, что веб-приложение будет постоянно загружено. При указании не использовать постоянное использование в случае простоя веб-приложения будут выгружены. Это позволяет экономить системные ресурсы. Так как разрабатываемый программный продукт является прототипом, то не используется параметр «всегда включено» – в этом нет необходимости.

Масштаб веб-приложений может быть повышен без необходимости повторного развертывания и написания кода. Для увеличения производительности и пропускной способности веб-приложения в Microsoft Azure необходимо изменить план службы приложений на более высокий на портале Azure и выполнить настройку.

План службы приложений – набор компонентов и ресурсов, которые используются и разделяются приложениями службы приложений, в том числе мобильные приложения, веб-приложения, приложения API, приложения логики. Для одного приложения службы приложений может быть выбран один план. Совместно использовать один план могут приложения, находящиеся в одном географическом регионе и использующие одну подписку. Приложения и планы службы приложений включаются в группу ресурсов, которая позволяет управлять совместно всеми приложениями. Как правило, в группу ресурсов определяются приложения, являющиеся частями одного проекта [33].

Приложение можно переметить в другой план службы приложений – новый или существующий. Также можно клонировать приложение в другой план службы приложений. При этом будет создана копия приложения. Данная опция особенно полезна при необходимости расположить приложение в другом регионе.

В зависимости от плана службы приложений может быть использовано ручное (бесплатный план, планы Shared и Basic) или автоматическое масштабирование (планы Standard и Premium). При ручном масштабировании в настройках указываем необходимое количество экземпляров. В таком случае число выбранных экземпляров не изменяется при изменении нагрузки.

Можно настроить автоматическое масштабирование на основе параметра нагрузки на CPU. На портале администрирования задаются минимальный и максимальный процент нагрузки на CPU, при которых происходит выделение или удаление экземпляров. Когда средняя нагрузка находится на максимальном значении, Microsoft Azure будет выделять новые экземпляры и балансировать нагрузку между ними. Когда же средняя нагрузка на CPU меньше минимального значения, экземпляры веб-приложения будут постепенно удалятся. Также можно указать допустимые пределы количества выделенных экземпляров. Тогда количество экземпляров будет всегда в данном диапазоне в независимости от других условий [34].

При действиях Azure по автомасштабированию администратору можно включить уведомления на электронную почту, кроме того можно указать веб-перехватчик – конечную точку HTTP/HTTPS, которая будет перенаправлять уведомления.

Другой вариант настройки автомасштабирования – правила расписания и производительности. В данном случае можно составить сложные правила автомасштабирования. На основе этих правил платформа Azure будет масштабировать приложение также самостоятельно. Параметры автомасштабирования могут быть настроены для ЦП, памяти, очереди диска, очереди HTTP и потока данных. При создании нового правила необходимо выбрать ресурс, из которого передается метрика. Это может быть тот же ресурс, который масштабируется, очередь хранилища или другой ресурс. Кроме того, для каждого правила настраиваются следующие переменные: оператор (равно, меньше чем и т. д.), пороговое значение метрики, длительность, агрегат времени, действие и значение, длительность ожидания для этого правила после предыдущего действия масштабирования. Для одного ресурса может быть применено несколько правил масштабирования.

Так как при разработке прототипа приложение не получает большой нагрузки, то для разработки использовался план Shared. Однако для конечной

ЭКЗЕМПЛЯР	ЯДРА	ОЗУ	ХРАНИЛИЩЕ	ЦЕНЫ
S1 Standard	1	1,75 ГБ	50 ГБ	$0,10/ч (~$74/мес.)
S2 Standard	2	3,50 ГБ	50 ГБ	$0,20/ч (~$149/мес.)
S3 Standard	4	7 ГБ	50 ГБ	$0,40/ч (~$298/мес.)
D1 Shared (предварительная версия)	Общая (240 минут ЦП в день)	0,5 ГБ	1 ГБ	~$0,013 за сайт в час (~$9,67/мес.)

Рис. 6. Сравнение планов

версии приложения необходима одна из версий standard (рис. 6).

Кроме того, могут быть настроены разные профили расписания, при которых масштабирование будет выполняться в определенное время и/или дни. [35] Так, можно снижать количество развернутых экземпляров на каникулах и повышать перед экзаменами. Данная функция не нужна при разработке прототипа, однако подразумевается при использовании конечной системы для внедрения.

Наиболее универсальный вариант для конечного приложения – standard s1 с настройками масштабирования на основе CPU и расписания с учетом каникул. Но нужно заметить, что точная конфигурация веб-приложения должна быть настроена для каждой школы индивидуально.

Масштабирование базы данных SQL

Подключенную к веб-приложению базу данных можно также масштабировать при необходимости. Для этого на портале Azure, аналогично веб-приложению, необходимо выбрать другую ценовую категорию [36].

Производительность базы данных SQL

Единица транзакций базы данных (DTU) — это единица измерения в базе данных SQL, представляющая относительную производительность баз данных в зависимости фактического показателя — количества транзакций базы данных [37]. DTU зависит от ценовой категории базы данных [azure]. Для разработки приложения выбрана база данных категории S0 Standard, ее характеристики отображены на рисунке 7. Для текущего этапа развития данного приложения не ожидается большого количества транзакций и размер базы данных в этом варианте достаточен.

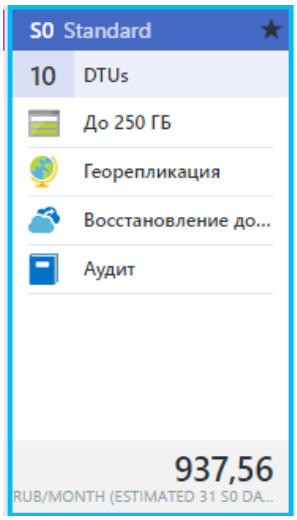


Рис. 7. План базы данных

3.1.3. Резервное копирование веб-приложений и базы данных

Для веб-приложений могут создаваться резервные копии. При этом резервные копии являются полной автономной копией приложения. Для возможности создания резервных копий план службы приложений должен быть «Standard» или «Premium». С помощью этих копий возможно восстановить предыдущее состояние веб-приложения или создать новое.

В резервную копию включается файлы и конфигурация веб-приложения, подключенные базы данных. Максимальный размер резервной копии – 10 Гб. Копии хранятся в контейнере и учетной записи хранения Azure. Они представляют собой ZIP-файлы с данными и XML-файл с манифестом для содержания архива. Базы данных хранятся в виде файла BACPAC (без расширения).

Резервные копии веб-приложений могут создаваться вручную или автоматически. Для автоматического создания резервных копий настраивается расписание. Создание копий вручную или настройка расписаний производится на портале Azure [38].

Если не нужна полная резервная копия веб-приложения, можно делать копию части приложения. Так как функции резервного копирования также выполняет репозиторий Git, с помощью которого производится непрерывное развертывание, то в полном регулярном копировании веб-приложения нет необходимости, но имеет смысл создавать частичную резервную копию – базы данных и конфигурации веб-приложения, которые не сохраняются в репозитории.

Кроме того, удобство использования резервных копий заключается в том, что это уже готовая рабочая версия приложения, и быстро можно восстановить сохраненную версию приложения, например, при проблемах с обновлением конечного продукта.

Следует отметить, что резервное копирование можно производить с помощью API REST [39], но в рамках данной работы этот метод не предлагается.

3.2. Логическая схема базы данных

База данных проектируется таким образом, чтобы можно было запрашивать вывод неуспевающих групп учеников и по связям между таблицами находить взаимосвязи между результатами обучения выбранных учащихся и учебным процессом. В результате можно выявить возможные причины неудовлетворительных результатов образовательного процесса.

Для решения поставленных задач в базе данных необходимо хранить следующие данные:

- Персональные данные учащихся и учителей (ФИО, дата рождения);
- Классы и список учеников;
- Должности учителей;
- История прохождений учителями курсов повышения квалификаций, прохождения аттестаций, получение наград, опыта работы, информация об образовании;
- Публикации, выполненные учениками и учителями (или только учителями);
- Предметы и темы предметов;
- Расписание уроков;
- Оценки за уроки, промежуточную аттестацию, итоговую аттестацию;
- Тестирования и задания тестирований, возможные баллы, полученные учеником баллы;
- Тип места поступления выпускника.

Диаграммы базы данных представлены на рисунках 8-11.

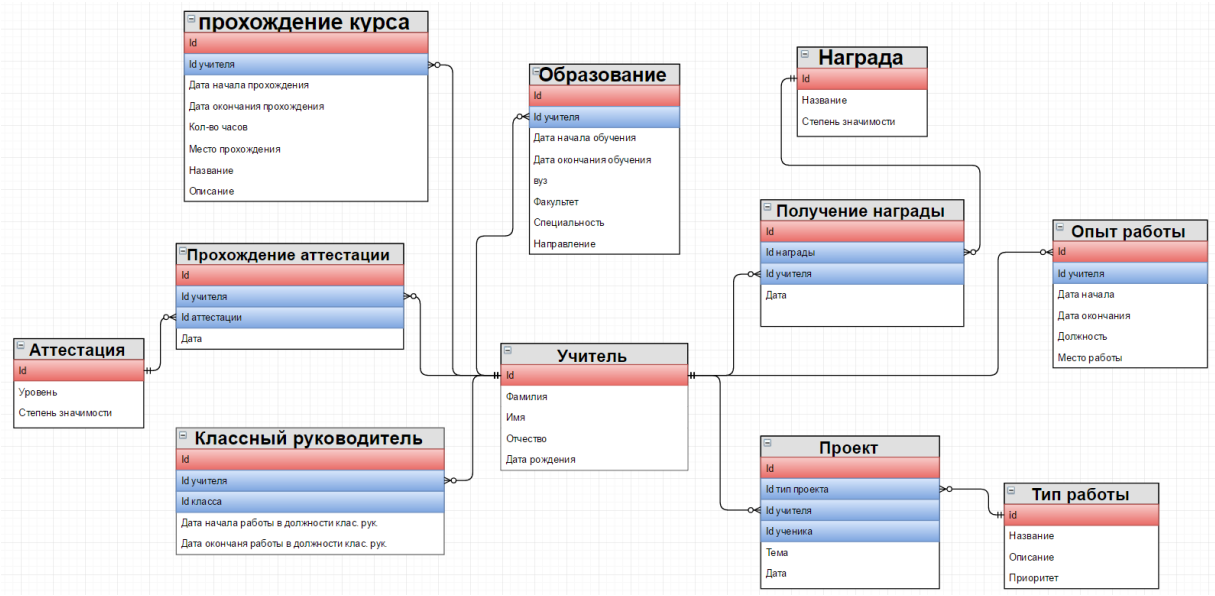


Рис. 8. Логическая схема БД, информация об учителе

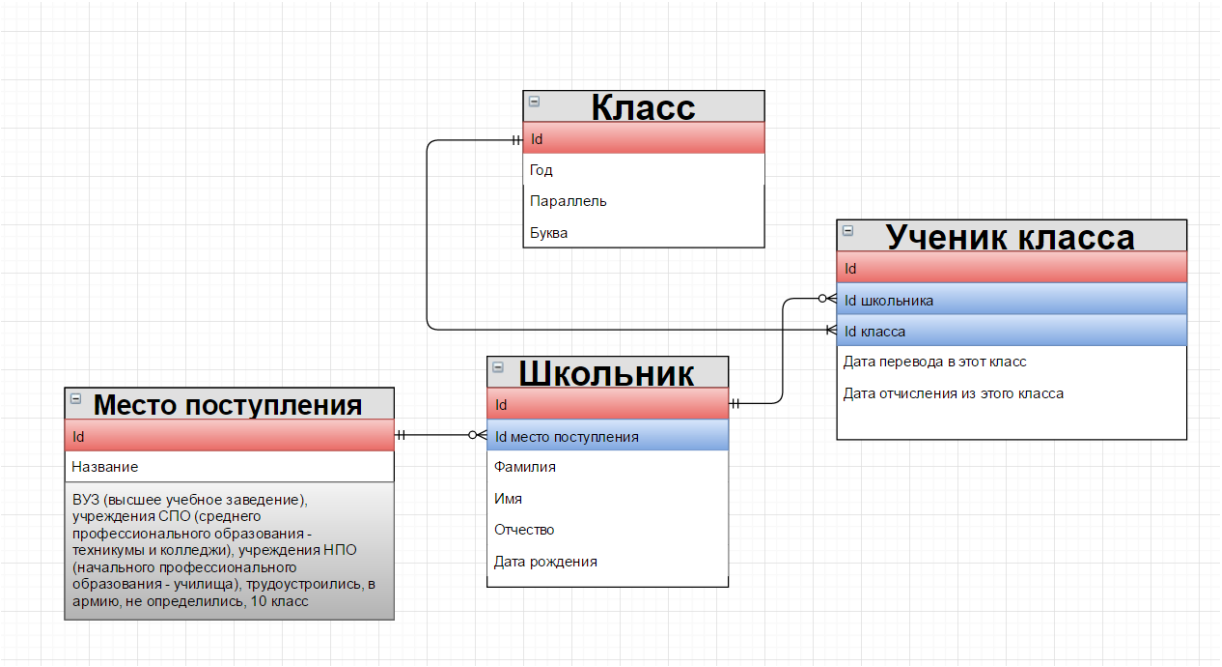


Рис. 9. Логическая схема БД. Информация об ученике

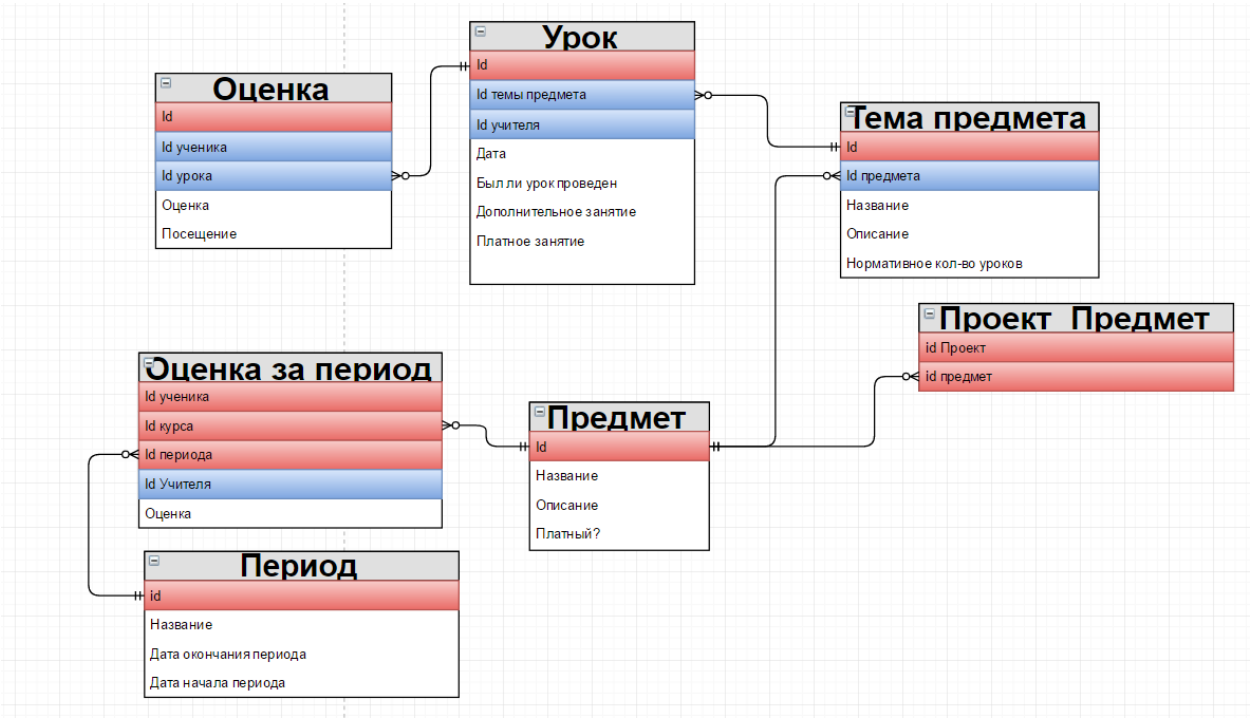


Рис. 10. Логическая схема БД. Оценки и уроки.

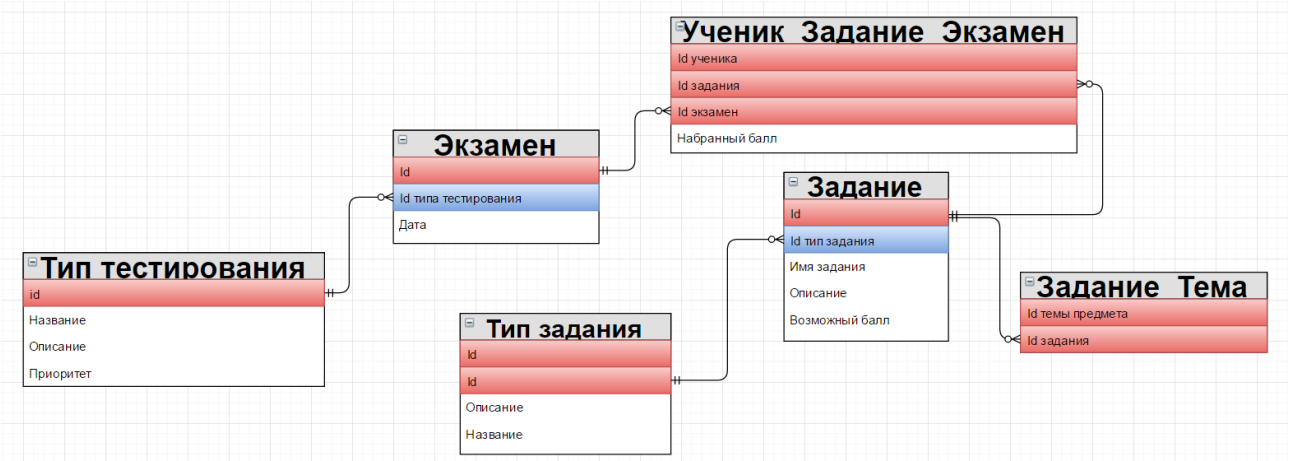


Рис. 11. Логическая схема БД. Экзамены и задания

3.3. Логическая архитектура

Как уже говорилось выше, было принято решение разработки прототипа автоматизированной системы мониторинга образовательного процесса в виде веб-приложения. Однако, предусматривается возможность перехода на десктоп-приложение по запросу заказчика. Так как в школах обычно установлена операционная система Windows 7, то было принято решение для

разработки приложений использовать стек технологий .NET Framework и язык программирования C#.

Разработана классическая трехуровневая архитектура (рис. 12). Данная архитектура подразумевает разделение программы на 3 части: уровень доступа к данным, уровень логики, уровень представления. Данный поход позволяет создавать более гибкие приложения.

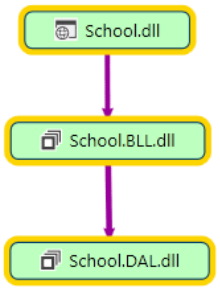


Рис. 12. Решение

Уровень представления (School) – проект ASP.NET MVC 5, включает в себя представления, компоненты пользовательского интерфейса, модели представлений, контроллеры, объекты контекста запроса.

Уровень бизнес-логики (School.BLL) – проект c# library, реализует основную логику приложения, связующее звено между уровнем представления и уровнем доступа к данным.

Уровень доступа к данным (School.DAL) – реализует операции с базой данных.

3.3.1. Уровень доступа к данным

Для доступа к данным используется Entity Framework 6 [40] – объектно-ориентированная технология для работы с данными, object-relation mapping решение для .NET Framework. Entity Framework (EF) 6 может использоваться с различными СУБД, в том числе – ySQL, Oracle, Progress, VistaDB, SQLite, Synergex, Firebird, Devart, OpenLink, SQL Anywhere, Sybase, и PostgreSQL (через поставщик Npgsql), поэтому решение с EF позволяет обеспечить определенную гибкость и сменить базу данных при необходимости, например, при желании заказчика.

Как известно, Entity Framework 6 предоставляет три возможных подхода для взаимодействия с базой данных: database first, model first, code first (иногда отдельно также выделяют code second – code first к существующей базе данных).

По причине того, что разрабатываемое приложение является прототипом и подразумевает дальнейшее развитие, а база данных проектируется с нуля, был выбран подход code first. Code first и code first migrations предоставляют удобный и простой способ управлением версиями базы данных. Так как при подходе code first схема базы данных полностью основывается на коде, то версия базы данных всегда будет соответствовать версии приложения, и контроль версий исходного кода будет покрывать контроль версий базы данных. К тому же, еще больше возможностей управления версиями разработчик получает благодаря миграциям, которые позволяют хранить различные версии моделей данных. Так как для школы потеря данных неприемлема, но будущие изменения схемы базы данных возможны, то при внесении изменений, не изменяющих критически структуру модели, благодаря миграциям возможно сохранение большей части данных. При изменении модели данных генерация классов, описывающих изменения, применяемые к структуре базы данных, происходит автоматически, то это значительно сокращает время разработки.

Для достижения еще большей гибкости используется LINQ to Entities – API-интерфейс LINQ для выполнения операций доступа к данным. LINQ to Entities вводит логическое отображение между концептуальной моделью Entity Framework и физической базой данных.

Доступ к данным реализуется с помощью паттернов репозиторий, что позволяет создать дополнительную абстракцию между уровнем доступа к данным и уровнем бизнес-логики, а также паттерн unit of work, основная цель которого – убедиться, что все репозитории используют один контекст данных, а также упростить работу со множеством репозиториев. Такое решение позволяет изолировать остальные уровни приложения от изменений в модели данных.

Специфические запросы для системы правил также находятся на уровне доступа к данным.

Часть исходного кода представлена в приложении 2.

Таким образом, взаимодействие с базой данных полностью реализовано в отдельном проекте – C# library.

3.3.2. Уровень бизнес-логики

Уровень бизнес-логики инкапсулирует всю основную логику приложения, передает данные с уровня доступа к данным на уровень представления и обратно, выполняет функции промежуточного уровня. Данный уровень реализован как отдельный проект c# library.

Для правильного разделения ответственности в архитектуре приложения уровень бизнес-логики работает с промежуточными сущностями. Использование библиотеки AutoMapper [41] позволяет упростить сопоставление классов из разных проектов приложения.

Согласно принципу инверсии управления, реализовано внедрение зависимостей. Данный подход обеспечивается с помощью использования IoC-контейнера Ninject [42]. Существует ряд различных IoC-контейнеров, но Ninject имеет версию для MVC 5, быстро устанавливается с помощью NuGet, не требует дополнительной настройки, имеет достаточно возможностей для установки дополнительных параметров и свойств, что хорошо подходит для разработки прототипа.

3.3.3. Уровень представления

Уровень представления – данный уровень отвечает только за формирование пользовательского интерфейса.

Проект уровня представления – ASP.NET MVC 5. Данная платформа представляет собой фреймворк, предназначенный для создания веб-приложений и реализующий паттерн MVC (model – view - controller). Благодаря использованию инфраструктуры MVC реализуется дополнительное

разделение ответственности, что облегчает сопровождение и масштабирование приложения в будущем, а также облегчает тестирование.

Как и в уровне бизнес-логики здесь используется IoC-контейнер Ninject для внедрения зависимостей и AutoMapper для сопоставления моделей представления с классами уровня бизнес-логики.

Большая часть фронт-энда создана с помощью синтаксиса движка представлений Razor. В отличие от формирования пользовательского интерфейса с помощью языка клиентской части JavaScript [43] данный подход позволяет быстрей создавать интерфейсы для отображения моделей представления, в том числе за счет использования включенных в ASP.NET MVC шаблонов представлений. Данные, получаемые в контроллерах с помощью AutoMapper приводятся к типу определяемых моделей представления и передаются в представление. Такой подходя является достаточным для разработки прототипа приложения.

Однако синтаксис движка представлений Razor все-таки обладает ограниченными возможностями в формировании пользовательского интерфейса. Было принято решение дополнительно использовать возможности языка JavaScript и библиотеки jQuery [44] для генерирования некоторых частей интерфейса, в частности модальных окон (рис. 13) и некоторых форм ввода. HTML-разметка формируется по мере необходимости, так неизвестно, понадобится ли ее формирование вообще, а фаза инициализации очень чувствительна к производительности. Возможно, это не так важно для разработки прототипа, но при дальнейшем развитии приложения имеет значение.

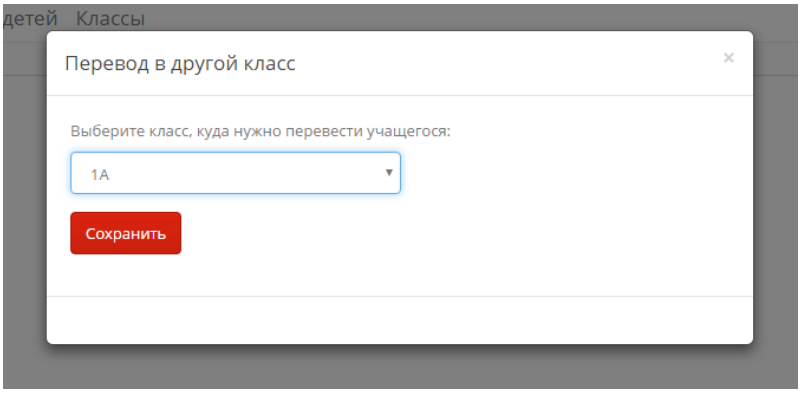


Рис. 13. Модальное окно

При заполнении форм задействуется механизм валидации с использованием атрибутов валидации при определении модели представления, хелперов валидации и библиотеки jquery.validate.unobtrusive. Благодаря этому некорректные данные не будут обрабатываться.

Технология AJAX применена с целью осуществления запросов к серверу без перезагрузки страницы. Для использования данного подхода в проект ASP.NET MVC включена библиотека jquery.unobtrusive-ajax.

С целью повышения скорости разработки и привлекательности интерфейса используется фреймворк bootstrap. Использование данного фреймворка обеспечивает совместимость с наиболее популярными браузерами. Кроме того, производится адаптивная верстка, что позволяет комфортно пользоваться сайтом с различных устройств.

Глава 4. Обзор работы приложения

4.1. Анализ показателей

На главной странице приложения отображается текущая статистика по учителям и предметам (рис. 14).

Статистика по предметам

Предмет	Успеваемость	Качество знаний
литературное чтение	100%	72%
окружающий мир	96%	70%
музыка	100%	100%
изобразительное искусство	100%	98%
физическая культура	100%	95%
немецкий язык	100%	0%

Рис. 14. Текущие показатели по предметам

Неудовлетворительные показатели: Анализировать

Показатель	Успеваемость	Качество знаний
немецкий язык	100%	0%
французский язык	99%	5%
русский язык	98%	52%
английский язык	97%	37%
У_21_Люс	98%	59%

Рис. 15. Неудовлетворительные показатели

Тут же отображаются неудовлетворительные показатели по предметам (рис. 15), учителям, по предмету и учителю вместе, если отдельно по данному предмету и учителю показатели удовлетворительные.

При нажатии на кнопку «Анализировать», которая находится над столбцом неудовлетворительных показателей, приложение производит анализ и выводит результат.

Фамилия	Имя	Предмет	Качество знаний	Успеваемость	Результаты анализа
	У_21_Леж	окружающий мир	46	78	Невыполнение плана учебной программы. Уроки не проведены в полной мере по темам: Животные; Птицы;
	У_21_Бар	математика	50	100	некомпетентность учителя в данном предмете

Рис. 16. Результаты анализа

На рисунке 16 видно, что программа вывела в первой строке результаты анализа – невыполнение плана учебной программы по темам животные и птицы. Данный результат был ожидаемым, так как в базу данных было загружено меньшее количество уроков, чем требуют того темы животные и птицы по плану. Каждой теме регламентировано определенное число уроков, и если по какой-то теме не проведено ожидаемое количество уроков, то срабатывает правило – невыполнение учебной программы.

4.2. Элементы электронного дневника

Для обеспечения работоспособности приложения реализованы необходимые функции электронного дневника – работа с основными компонентами школьного образовательного процесса: учениками (рис. 17, 18), классами, учителями, уроками, оценками за периоды, экзамены.

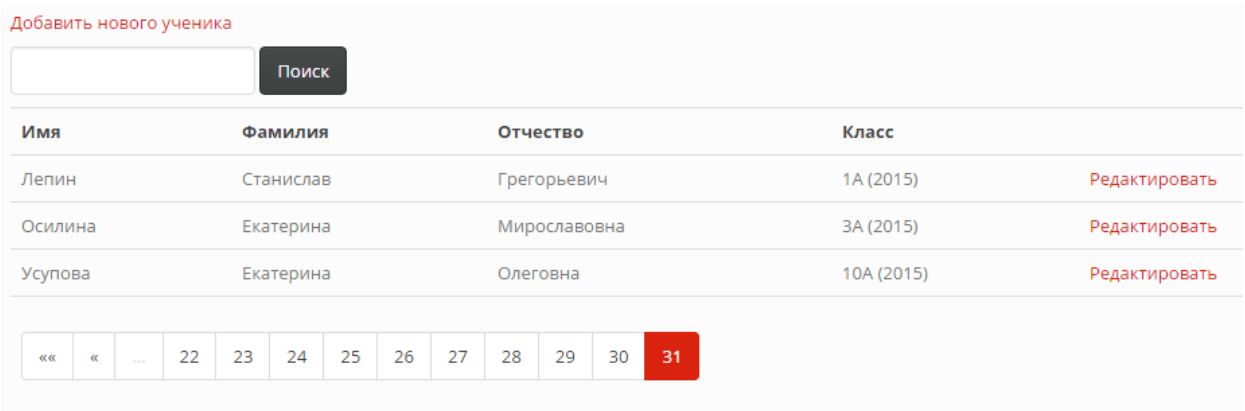


Рис. 17. Ученики

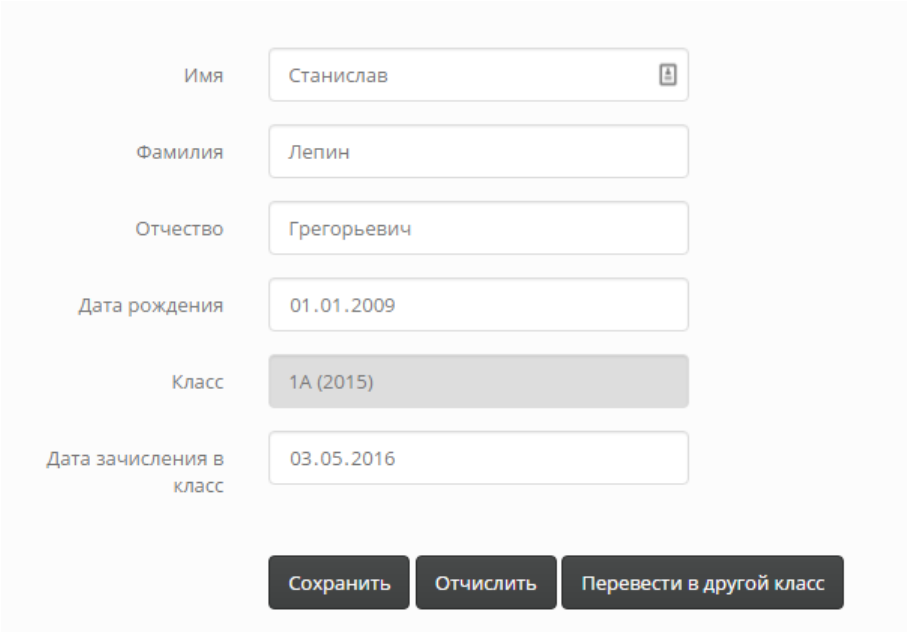


Рис. 18. Редактирование данных ученика

4.3. Конструирование правил

В разработке описываемого приложения учитывается, что у конечного пользователя может появиться необходимость в изменении существующих правил и добавлении новых. Данная возможность предоставляется с помощью страницы настройки правил.

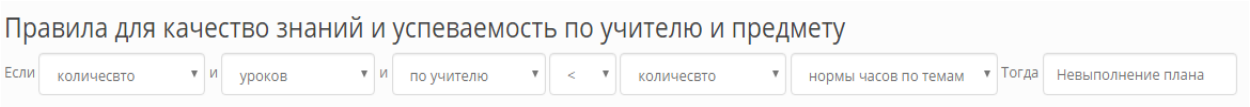


Рис. 19. Пример задания правила

Для каждого анализируемого на данный момент показателя качества результатов обучения пользователь может задавать свои правила выводов. Для правила вывода задается сравниваемый показатель (качество знаний, успеваемость, посещаемость, количество и т. д.), объекты, для которых должен высчитываться показатель (уроки, учителя, предметы, темы и т. д.), функция сравнения, числовое значения для сравнения либо показатель в виде функции от других объектов. Каждому правилу пользователь может сопоставлять причины низких показателей качества образования.

Объекты выбираются в обобщенном виде – тогда для каждого неудовлетворительного показателя берутся те объекты, по которым был подсчитан данный показатель. Пример построения правила представлен на рис. 19.

Конечно, возможные варианты правил ограничены реализованными в коде методами получения информации. Для расширения подобного функционала необходима работа с экспертами и добавление необходимых им функций.

Для всех показателей настраивается критическое значение.

Таким образом, обеспечивается большая гибкость приложения и дополнительные удобства использования для конечного пользователя.

4.4. Тестирование производительности

Тестирование производительности с нагрузкой было произведено на портале управления Azure.

Результаты тестирования представлены на рисунках 20-23.

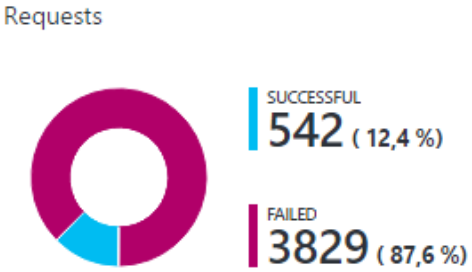


Рис. 2014. 30 пользователей. План Shared

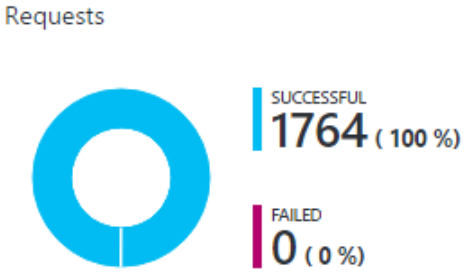


Рис.21. 30 пользователей. План Standard s1, автомасштабирование до 5 экземпляров

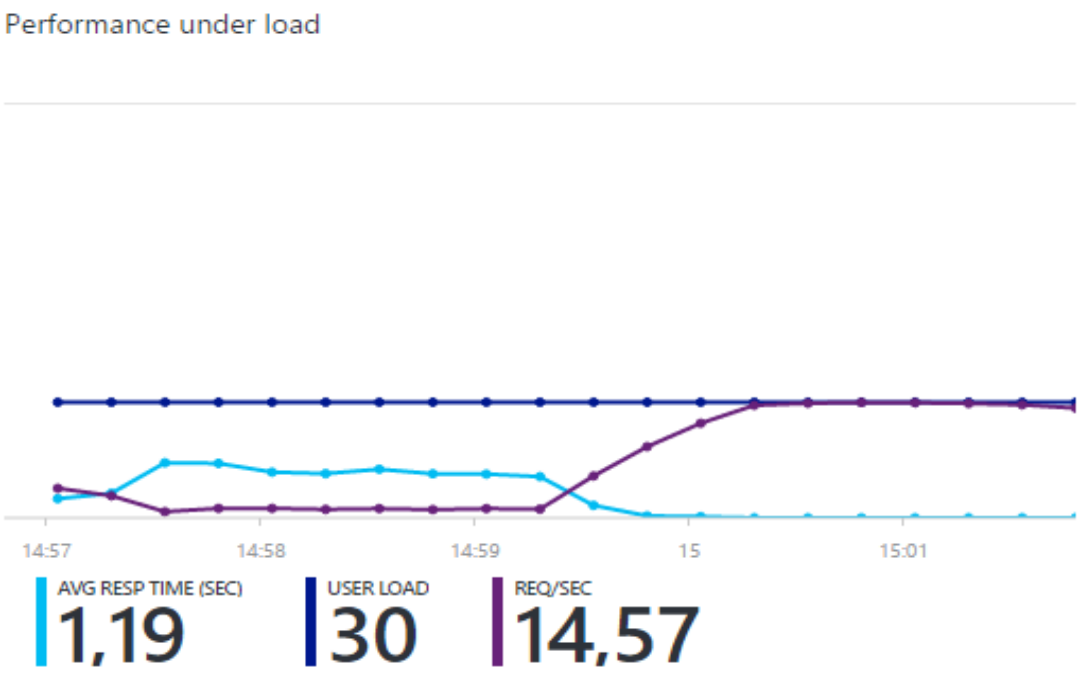


Рис. 22. 30 пользователей. План Shared

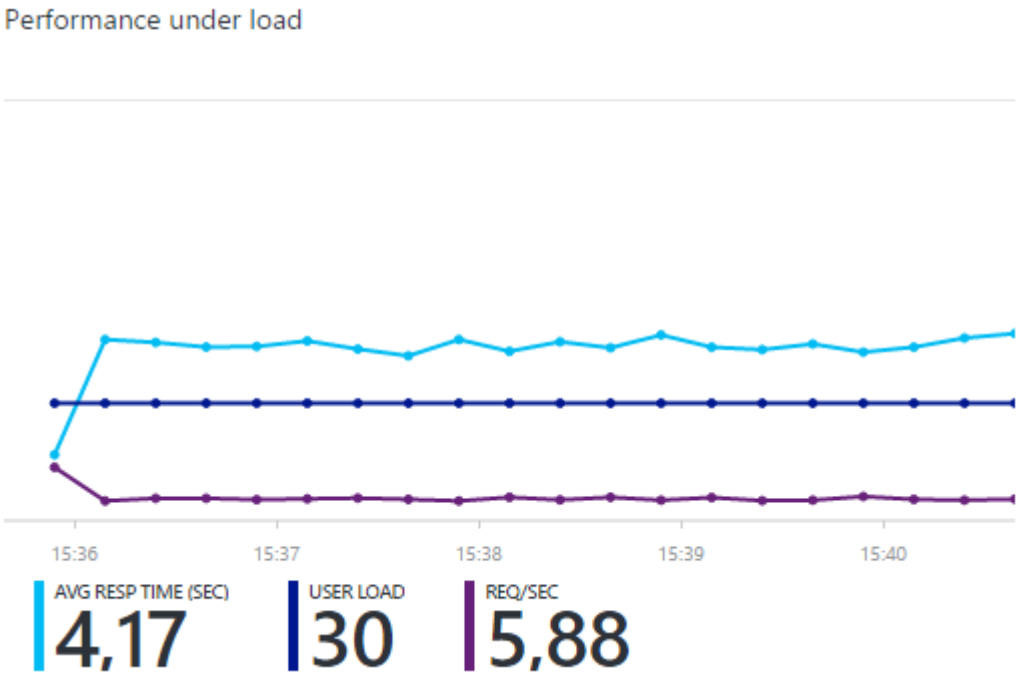


Рис. 23. 30 пользователей. План Standard с автомасштабированием

Результаты тестирования подтверждают тот факт, что план shared подходит только для разработки и тестирования, либо при низкой нагрузке приложения. Показатели для плана standard и прототипа приложения можно считать удовлетворительными.

Выводы

В рамках данной работы была поставлена цель – повышение качества процесса мониторинга образовательного процесса в школах. Для достижения данной цели были поставлены задачи составления системы правил и разработки прототипа программного обеспечения, автоматизирующего использование обозначенных правил.

С учетом существующих решений и их недостатков, посредством самостоятельного изучения предметной области и помощи экспертов в данной области был составлен обширный ряд правил и предложена возможность их автоматизированной реализации с использованием реляционных баз данных.

Было разработано программное обеспечение с использованием инфраструктуры Microsoft Azure для размещения данного приложения. В частности, реализовано веб-приложение службы приложений с базой данных Azure SQL. Было проведено тестирование с нагрузкой работы приложения на базе облачной инфраструктуры с удовлетворительными результатами.

Так как разработанное приложение является прототипом, то оно не учитывает закон РФ «О персональных данных», согласно которому персональные данные пользователей должны хранится на серверах, расположенных на территории РФ. В то же время у Microsoft Azure таких серверов нет. В связи с этим необходимо подробней изучить закон и принять архитектурные решения, которые позволят избежать нарушения законодательства. Например, такими решениями могут быть: размещение персональных данных в зашифрованном виде и расшифровка их с помощью программного обеспечения, не нарушающего названный закон, построение гибридной архитектуры с передачей в облако Azure обезличенных данных, отказ от использования услуг Azure.

Реализована возможность конструирования правил пользователем и покрыты наиболее часто применяющиеся правила. Одно из возможных

развитий приложения – расширение возможностей конструирования правил, для этого требуется сотрудничество с экспертами в данной области.

Другое возможное развитие приложения – при использовании данной системы на большом количестве данных возможно использовать полученные результаты анализа для расширения возможностей сопоставления неудовлетворительных показателей обучения и их причин с использованием алгоритмов машинного обучения.

Заключение

С целью повышения качества процесса мониторинга образовательного процесса в школах была разработана система правил для поиска потенциальных причин проблем в обучении.

Разработан прототип приложения с использованием разработанной системы правил, который позволяет автоматизировать поиск возможных причин проблем в образовательном процессе.

Система для внутришкольной оценки качества образования может быть удобным инструментом для повышения уровня предоставляемых образовательных услуг. Данные функции позволят администрации школ, методическим объединениям учителей значительно снизить время анализа результатов обучения, сфокусироваться на решении существующих проблем и получать актуальную информацию о текущих образовательных показателях в любой момент времени. Кроме того, хранение данных в электронном виде позволит значительно снизить количество бумажной работы сотрудникам школы.

Список Литературы

1. Качество образования – Министерство образования. http://studydoc.ru/doc/728809/kachestvo-obrazovaniya---ministerstvo-obrazovaniya
2. Федеральный закон об образовании в РФ. Статья 28. Пункты 10-13. http://zakon-ob-obrazovanii.ru/28.html
3. «Формирование школьной системы оценки качества образования». Сборник методических материалов. http://rsoko.orcoko.ru/pages/info
4. Федеральный закон об образовании в РФ. Статья 97. http://zakon-ob-obrazovanii.ru/97.html
5. Приказ Министерства образования и науки Российской Федерации от 15 января 2014 г. № 14 «Об утверждении показателей мониторинга системы образования». http://xn--273--84d1f.xn--p1ai/akty_minobrnauki_rossii/prikaz-minobrnauki-rf-ot-15012014-no-14
6. Постановление Правительства Российской Федерации от 5 августа 2013 г. N 662 г. Москва «Об осуществлении мониторинга системы образования». http://rg.ru/2013/08/19/monitoring-site-dok.html
7. «ЭлЖур». Электронный журнал для школы. http://eljur.ru/vse-funkcii-elektronnogo-zhurnala-dlya-shkol
8. «SmileS.Школьная карта». https://www.shkolnaya-karta.ru/
9. «Дневник.ру». https://dnevnik.ru/
10. «ИРТех». Модуль «МСОКО». http://www.ir-tech.ru/?products=msoko
11. Чванова М. С., Киселева И. А., Молчанов А. А., Бозюкова А. Н. Использование экспертных систем в образовании // Гаудеамус, 2013. Вып. № 1 (21). С. 47–54.
12. Касымалиева А. Т. Автоматизированная информационная экспертная система проведения SWOT-анализа вуза // Бизнес-информатика, 2014. №2(28). С. 72-77.
13. Экспертная система оценки рисков в области ИТ-образования. http://elib.osu.ru/bitstream/123456789/487/1/3301-3306.pdf
14. Приказ Министерства образования и науки Российской Федерации от 1 июля 2013 г. N 499 г. Москва «Об утверждении Порядка организации и осуществления образовательной деятельности по

дополнительным профессиональным программам». http://rg.ru/2013/08/28/minobr-dok.html

15. Microsoft Azure. https://azure.microsoft.com/ru-ru/
16. Введение в Microsoft Azure. https://azure.microsoft.com/ru-ru/documentation/articles/fundamentals-introduction-to-azure/
17. .NET. https://www.microsoft.com/net
18. Node.js https://nodejs.org/en/
19. PHP. http://php.net/
20. Python. https://www.python.org/
21. Java. http://www.oracle.com/technetwork/java/index.html
22. Официальная документация C#. https://msdn.microsoft.com/ru-ru/library/kx37x362.aspx
23. ASP.NET MVC. http://www.asp.net/mvc
24. Портал управления Azure. https://portal.azure.com/
25. Bitbucket. https://bitbucket.org/
26. Github. https://github.com/
27. VSTS. https://www.visualstudio.com/ru-ru/products/visual-studio-team-services-vs.aspx
28. Git. https://git-scm.com/
29. Mercurial. https://www.mercurial-scm.org/
30. Azure SQL. https://azure.microsoft.com/ru-ru/services/sql-database/
31. Официальная документация Microsoft SQL Server. https://msdn.microsoft.com/ru-ru/library/bb545450.aspx
32. Официальная документация SQL Server Management Studio. https://msdn.microsoft.com/ru-ru/library/ms174173.aspx
33. Подробный обзор планов службы приложений Azure. https://azure.microsoft.com/ru-ru/documentation/articles/azure-web-sites-web-hosting-plans-in-depth-overview/
34. Автоматическое масштабирование веб-сайтов, облачных сервисов и виртуальных машин. https://msdn.microsoft.com/ru-ru/dn383629.aspx
35. Масштабирование числа экземпляров. https://azure.microsoft.com/ru-ru/documentation/articles/insights-how-to-scale/

36. Масштабирование веб-приложения в службе приложений Azure. https://azure.microsoft.com/ru-ru/documentation/articles/web-sites-scale/

37. Введение в базы данных SQL. https://azure.microsoft.com/ru-ru/documentation/articles/sql-database-technical-overview/

38. Резервное копирование веб-приложений в службе приложений Azure. https://azure.microsoft.com/ru-ru/documentation/articles/web-sites-backup/

39. Использование REST для резервного копирования и восстановления приложений службы приложений. https://azure.microsoft.com/ru-ru/documentation/articles/websites-csm-backup/

40. Официальная документация Entity Framework. https://msdn.microsoft.com/en-us/data/ee712907.aspx

41. AutoMapper. http://automapper.org/

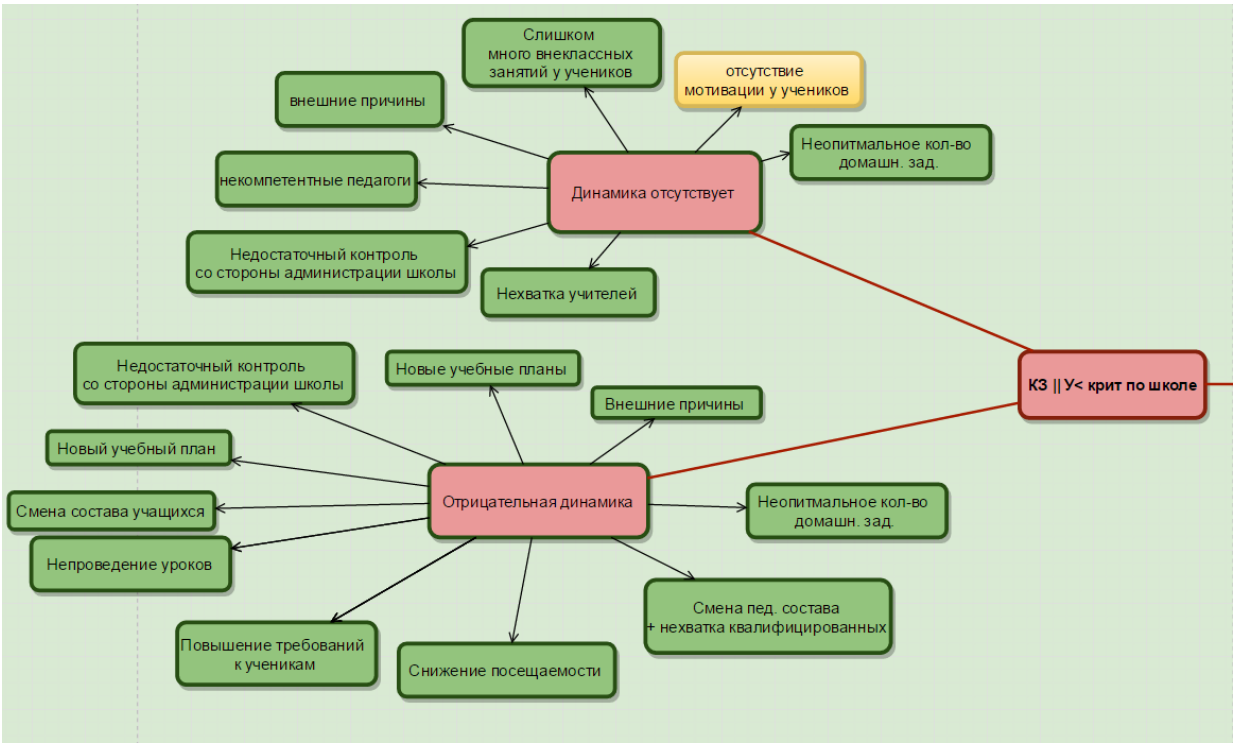
42. Ninject. http://www.ninject.org/

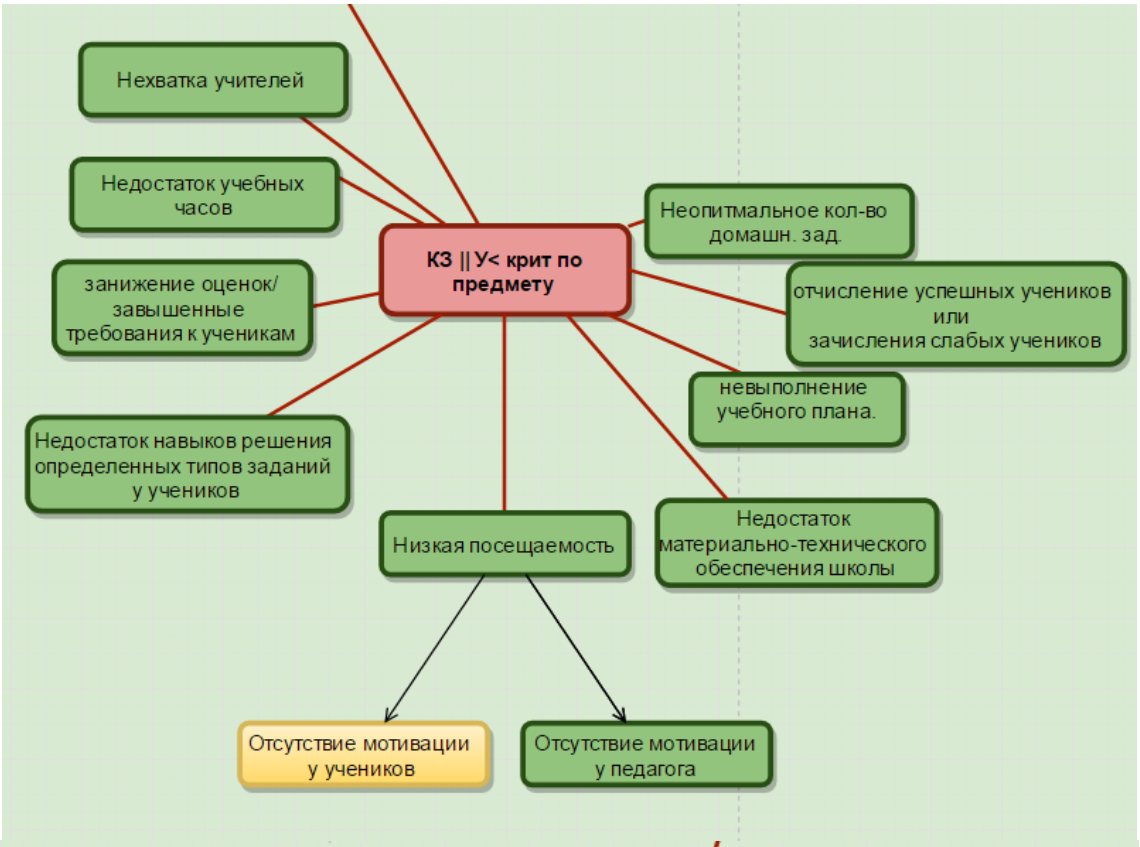
43. JavaScript. https://www.javascript.com/

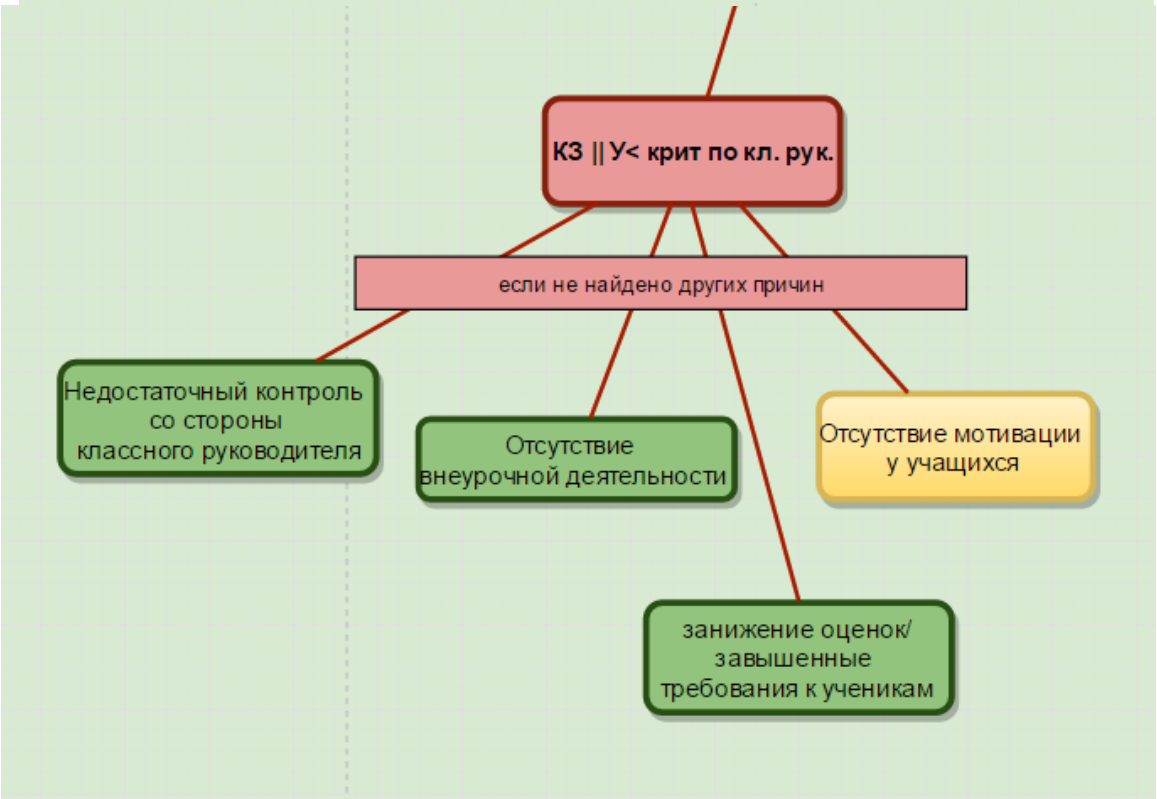
44. JQuery. https://jquery.com/

Приложения

Приложение 1. Схемы сопоставления проблем и причин




Нехватка учителей
Недостаток учебных часов
занижение оценок/ завышенные требования к ученикам
КЗ || У< крит по предмету
Неопитмальное кол-во домашн. зад.
отчисление успешных учеников или зачисления слабых учеников
невыполнение учебного плана.
Недостаток навыков решения определенных типов заданий у учеников
Низкая посещаемость
Недостаток материально-технического обеспечения школы
Отсутствие мотивации у учеников
Отсутствие мотивации у педагога


КЗ || У< крит по кл. рук.
если не найдено других причин
Недостаточный контроль со стороны классного руководителя
Отсутствие внеурочной деятельности
Отсутствие мотивации у учащихся
занижение оценок/ завышенные требования к ученикам

Приложение 2. Исходный код

Исходный код приложения слишком велик, чтобы представить его полностью.

```
using System;
using System.Collections.Generic;
using System.Linq;
using School.DAL.Repositories;

namespace School.DAL.Interfaces
{
    public interface IRepository<T> where T: class
    {
        IEnumerable<T> GetAll();
        T GetById(int id);
        IEnumerable<T> Where(Func<T, Boolean> func);
        void Create(T obj);
        void Update(T obj);
        void Delete(T obj);
        QueryConstructor<T> Query();
        IQueryable<T> Queryable();
        IRepository<TT> GetRepository<TT>() where TT : class;
    }
}
```

```
using System;

namespace School.DAL.Interfaces
{
    public interface IUnitOfWork : IDisposable
    {
        void Save();
        IRepository<T> Repository<T>() where T : class;

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Linq.Expressions;
using School.DAL.EF;
using School.DAL.Interfaces;

namespace School.DAL.Repositories
{
    public class Repository<T>: IRepository<T> where T: class
    {
        private SchoolContext db;
        private DbSet<T> _dbSet;
        private IUnitOfWork _unitOfWork;
```

```
public Repository(SchoolContext context, IUnitOfWork unitOfWork)
{
    db = context;
    _dbSet = db.Set<T>();
    _unitOfWork = unitOfWork;
}

public virtual IEnumerable<T> GetAll()
{
    return _dbSet.AsEnumerable();
}

public virtual IEnumerable<T> Where(Func<T, bool> func)
{
    return _dbSet.Where(func).AsEnumerable();
}

public virtual void Create(T obj)
{
    if (obj == null)
    {
        throw new ArgumentNullException(nameof(obj));
    }
    _dbSet.Add(obj);
}

public virtual void Update(T obj)
{
    if (obj == null)
    {
        throw new ArgumentNullException(nameof(obj));
    }
    db.Entry(obj).State = EntityState.Modified;

}

public virtual void Delete(T obj)
{
    if (obj == null)
    {
        throw new ArgumentNullException(nameof(obj));
    }
    _dbSet.Remove(obj);
}
public virtual T GetById(int id)
{
    T obj = _dbSet.Find(id);
    if (obj == null)
    {
        throw new ArgumentNullException(nameof(obj));
    }
    return obj;
}

public virtual QueryConstructor<T> Query()
{
    var queryConstructor =
        new QueryConstructor<T>(this);

    return queryConstructor;
}

internal IQueryable<T> Get(
Expression<Func<T, bool>> filter = null,
```

```
        Func<IQueryable<T>,
            IOrderedQueryable<T>> sortBy = null,
        List<Expression<Func<T, object>>>
            includeEntities = null,
        int? page = null,
        int? pageSize = null)
        {
            IQueryable<T> query = _dbSet;

            if (includeEntities != null)
                includeEntities.ForEach(i => { query = query.Include(i); });

            if (filter != null)
                query = query.Where(filter);

            if (sortBy != null)
                query = sortBy(query);

            if (page != null && pageSize != null)
                query = query
                    .Skip((page.Value - 1) * pageSize.Value)
                    .Take(pageSize.Value);

            return query;
        }

        public IQueryable<T> Queryable()
        {
            return _dbSet;
        }

        public IRepository<TT> GetRepository<TT>() where TT : class
        {
            return _unitOfWork.Repository<TT>();
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace School.DAL.Repositories
{
    public sealed class QueryConstructor<T> where T : class
    {
        private readonly List<Expression<Func<T, object>>>
            _includeEntities;

        private readonly Repository<T> _repository;
        private Expression<Func<T, bool>> _whereFilter;
        private Func<IQueryable<T>,
            IOrderedQueryable<T>> _sortBy;
        private int? _page;
        private int? _pageSize;

        public QueryConstructor(Repository<T> repository)
        {
            _repository = repository;
            _includeEntities =
                new List<Expression<Func<T, object>>>();
```

```
        }

        public QueryConstructor<T> Filter(
            Expression<Func<T, bool>> expression)
        {
            _whereFilter = expression;
            return this;
        }

        public QueryConstructor<T> SortBy(
            Func<IQueryable<T>, IOrderedQueryable<T>> sortBy)
        {
            _sortBy = sortBy;
            return this;
        }

        public QueryConstructor<T> Include(
            Expression<Func<T, object>> expression)
        {
            _includeEntities.Add(expression);
            return this;
        }

        public IEnumerable<T> GetPage(
            int page, int pageSize, out int totalCount)
        {
            _page = page;
            _pageSize = pageSize;
            totalCount = _repository.Get(_whereFilter).Count();

            return _repository.Get(
                _whereFilter,
                _sortBy, _includeEntities, _page, _pageSize);
        }

        public IEnumerable<T> Get()
        {
            return _repository.Get(
                _whereFilter,
                _sortBy, _includeEntities, _page, _pageSize);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.Entity.Validation;
using School.DAL.EF;
using School.DAL.Interfaces;

namespace School.DAL.Repositories
{
    public class WrapperContext: IUnitOfWork
    {
        private SchoolContext db;
        private Dictionary<string, object> _repositories;
        private bool disposed = false;

        public WrapperContext(string connectionString)
        {
            db = new SchoolContext(connectionString);
```

```
        }

        public IRepository<T> Repository<T>() where T : class
        {
            if (_repositories == null)
            {
                _repositories = new Dictionary<string, object>();
            }

            var type = typeof (T).Name;
            if (!_repositories.ContainsKey(type))
            {
                var repositoryType = typeof (Repository<>);
                var repositoryInstance =
Activator.CreateInstance(repositoryType.MakeGenericType(typeof (T)), db, this);
                _repositories.Add(type, repositoryInstance);
            }
            return (IRepository<T>) _repositories[type];
        }


        public void Save()
        {
            try
            {
                db.SaveChanges();
            }
            catch (DbEntityValidationException dbEx)
            {
                string errorMessage = string.Empty;
                foreach (var validationErrors in dbEx.EntityValidationErrors)
                {
                    foreach (var validationError in validationErrors.ValidationErrors)
                    {

                        errorMessage += Environment.NewLine +
                                        $"Property: {validationError.PropertyName} Error:
{validationError.ErrorMessage}";
                    }
                }
                throw new Exception(errorMessage, dbEx);
            }
        }

        public virtual void Dispose(bool disposing)
        {
            if (!disposed)
            {
                if (disposing)
                {
                    db.Dispose();
                }
                disposed = true;
            }
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using School.DAL.Entities;
using School.DAL.Interfaces;

namespace School.DAL.Repositories
{
    public static class GradeForTermRepository
    {
        public static IEnumerable<object> GetPercentOfGradesForTermByTeacherAndSubject(
            this IRepository<GradeForTerm> gradeRepository,
            IEnumerable<int> grades, int termId)
        {
            var query = gradeRepository.Queryable()
                .Where(h => h.TermId == termId).GroupBy(f => new { f.SubjectId,
f.TeacherId })
                .Select(x => new {x.Key.SubjectId, x.Key.TeacherId , Percent = x.Count(p
=> grades.Contains(p.Grade)) * 100 / x.Count() }).ToList();
            return query;
        }

        public static IEnumerable<object> GetPercentOfGradesForTerm(
            this IRepository<GradeForTerm> gradeRepository,
            IEnumerable<int> grades, Func<GradeForTerm, int> groupBy, int termId, int
studyYear)
        {
            var query = gradeRepository.Queryable()
                .Include(g => g.Pupil)
                .Where(h => h.TermId == termId && h.Pupil.Enrolment.Value.Year ==
studyYear).GroupBy(groupBy)
                .Select(x => new { x.Key, Percent = x.Count(p =>
grades.Contains(p.Grade)) * 100 / x.Count() }).ToList();
            return query;
        }

        public static int PercentValueOfPupilsByOtherTeachersAndSubjects(
            this IRepository<GradeForTerm> repository, int teacherId, int subjectId, int
termId,
            IEnumerable<int> pupilIds, IEnumerable<int> searchGrades)
        {
            var grades = repository.Queryable();
            var resultGrades = from grade in grades
            where grade.TeacherId != teacherId
                  && grade.SubjectId != subjectId
                  && grade.TermId == termId
                  && pupilIds.Contains(grade.PupilId)
            select grade.Grade;

            var percentValue = resultGrades.Any() ? resultGrades.Count(p =>
searchGrades.Contains(p))*100/resultGrades.Count() : 100;
            return percentValue;
        }

        public static IEnumerable<int> GetPupilsByGrades(this IRepository<GradeForTerm>
repository,
            int teacherId, int subjectId, IEnumerable<int> grades, int termId)
        {
            var allGrades = repository.Queryable();
            var pupilIds =
                from grade in allGrades
```

```
                where grade.TermId == termId
                      && grade.TeacherId == teacherId
                      && grade.SubjectId == subjectId
                      && grades.Contains(grade.Grade)
                select grade.PupilId;
            return pupilIds.ToList();
        }


    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using School.DAL.Entities;
using School.DAL.Interfaces;

namespace School.DAL.Repositories
{
    public static class LessonRepository
    {
        public static IEnumerable<int> GetTeachers(this IRepository<Lesson> repository,
            int subjectId, Term term, IEnumerable<int> pupilIds)
        {
            var lessons = repository.Queryable();
            var marks = repository.GetRepository<Mark>().Queryable();
            var topics = repository.GetRepository<Topic>().Queryable();

            var result =
                from mark in marks
                join lesson in lessons
                    on mark.LessonId equals lesson.Id
                join topic in topics on lesson.TopicId equals topic.Id
                where topic.SubjectId == subjectId
                      && lesson.DateTime.CompareTo(term.Start) >= 0 &&
lesson.DateTime.CompareTo(term.End) <= 0
                      && !lesson.Extra && !lesson.FeeBased
                      && pupilIds.Contains(mark.PupilId)
                group lesson by lesson.TeacherId
                into gr
                select gr.Key;

            return result.ToList();
        }


        public static int PupilsAttendance(
            this IRepository<Lesson> repository, int teacherId, int subjectId,
IEnumerable<int> pupilIds, Term term)
        {
            var lessons = repository.Queryable();
            var marks = repository.GetRepository<Mark>().Queryable();
            var topics = repository.GetRepository<Topic>().Queryable();

            var result =
                (from mark in marks
                join lesson in lessons
                    on mark.LessonId equals lesson.Id
                join topic in topics on lesson.TopicId equals topic.Id
                 where topic.SubjectId == subjectId
                 && lesson.TeacherId == teacherId
```

```
                && lesson.DateTime.CompareTo(term.Start) >= 0 &&
lesson.DateTime.CompareTo(term.End) <= 0
                && !lesson.Extra && !lesson.FeeBased
                && pupilIds.Contains(mark.PupilId)
                select mark.Attendance).ToList();

            int att;
            if (result.Any())
            {
                att = result.Count(i => i.Equals(true)) * 100 / result.Count();
            }
            else
            {
                att = 100;
            }

            return att;

        }

        public static int PupilsAttendanceEcxeptTeacherAndSubject(
            this IRepository<Lesson> repository, int teacherId, int subjectId,
IEnumerable<int> pupilIds, Term term)
        {
            var lessons = repository.Queryable();
            var marks = repository.GetRepository<Mark>().Queryable();
            var topics = repository.GetRepository<Topic>().Queryable();

            var result =
                (from mark in marks
                    join lesson in lessons
                        on mark.LessonId equals lesson.Id
                    join topic in topics on lesson.TopicId equals topic.Id
                    where topic.SubjectId != subjectId
                          && lesson.TeacherId != teacherId
                          && lesson.DateTime.CompareTo(term.Start) >= 0 &&
lesson.DateTime.CompareTo(term.End) <= 0
                          && !lesson.Extra && !lesson.FeeBased
                          && pupilIds.Contains(mark.PupilId)
                    select mark.Attendance).ToList();

            int att;
            if (result.Any())
            {
                att = result.Count(i => i.Equals(true))*100/result.Count();
            }
            else
            {
                att = 0;
            }

            return att;

        }

        public static int GetPupilsMarksCount(
            this IRepository<Lesson> repository,
            int subjectId, Term term, IEnumerable<int> pupilIds )
        {
            var lessons = repository.Queryable();
            var marks = repository.GetRepository<Mark>().Queryable();
            var topics = repository.GetRepository<Topic>().Queryable();

            var result =
```

```
                (from mark in marks
                    join lesson in lessons
                        on mark.LessonId equals lesson.Id
                    join topic in topics on lesson.TopicId equals topic.Id
                    where topic.SubjectId == subjectId
                          && lesson.DateTime.CompareTo(term.Start) >= 0 &&
lesson.DateTime.CompareTo(term.End) <= 0
                          && pupilIds.Contains(mark.PupilId)
                    select mark.Id).Count();


            return result;



        }



        public static int GetLessonsQuantityOfTeacher(this IRepository<Lesson>
repository, int teacherId, Term term)
        {
            var lessons =
                repository.Queryable()
                    .Count(
                        p =>
                            p.TeacherId == teacherId && p.DateTime.CompareTo(term.Start)
>= 0 &&
                            p.DateTime.CompareTo(term.End) <= 0 && p.WasHeld);
            return lessons;
        }

    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using School.DAL.Entities;
using School.DAL.Interfaces;

namespace School.DAL.Repositories
{
    public static class TopicRepository
    {
        public static IEnumerable<Topic> GetUnimplementTopics(this IRepository<Topic>
repository,
            int teacherId, int subjectId, Term term, IEnumerable<int> pupilIds)
        {

            var topics = repository.Queryable();
            var lessons = repository.GetRepository<Lesson>().Queryable();
            var marks = repository.GetRepository<Mark>().Queryable();


            var missedTopics = from topic in topics
                join lesson in lessons
                    on topic.Id equals lesson.TopicId into tGroup
                from item in tGroup.DefaultIfEmpty()
                join mark in marks
                on item.Id equals mark.LessonId
                where (topic.SubjectId == subjectId && !item.Extra
                       && !item.FeeBased && item.WasHeld && item.TeacherId == teacherId
```

```
                        && item.DateTime.CompareTo(term.Start) >= 0 &&
item.DateTime.CompareTo(term.End) <= 0
                        && topic.Lessons.Count < topic.RequiredQuantityLessons
                        && pupilIds.Contains(mark.PupilId)) || item.TopicId == null
                    select topic;
            var t = missedTopics.Distinct().ToList();

            return t;
        }

    }
}
```

```
    using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using School.BLL.Infrastructure;
using School.BLL.Interfaces;
using School.DAL.Interfaces;
using static School.BLL.Infrastructure.AutoMapperConfig;

namespace School.BLL.Services
{
    public abstract class EntityService<T, TD>: IEntity<TD> where TD: class where T:class
    {
        IUnitOfWork Database { get; set; }

        protected EntityService(IUnitOfWork unitOfWork)
        {
            Database = unitOfWork;
        }

        public virtual IEnumerable<TD> GetAll()
        {
            return Mapper.Map<IEnumerable<TD>>(Database.Repository<T>().GetAll());
        }

        public virtual TD FindById(int? id)
        {
            if(id == null)
                throw new ValidationException("Id равен 0", "");
            return Mapper.Map<TD>(Database.Repository<T>().GetById(id.Value));
        }

        public virtual void Create(TD obj)
        {
            Database.Repository<T>().Create(Mapper.Map<T>(obj));
            Database.Save();
        }

        public virtual void Update(TD obj)
        {
            Database.Repository<T>().Update(Mapper.Map<T>(obj));
            Database.Save();
        }

        public virtual void Dispose()
        {
            Database.Dispose();
        }

        public virtual IEnumerable<T> Where(Expression<Func<T, bool>> exp)
        {
            return Database.Repository<T>().Query().Filter(exp).Get();
```

```
        }
        protected virtual void Delete(int id)
        {
            var entity = FindById(id);
            Database.Repository<T>().Delete(Mapper.Map<T>(entity));
            Database.Save();
        }

        protected virtual void Delete(TD obj)
        {
            Database.Repository<T>().Delete(Mapper.Map<T>(obj));
            Database.Save();
        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using School.BLL.DTO;
using School.BLL.Interfaces;
using School.DAL.Interfaces;
using School.BLL.Infrastructure;
using School.DAL.Entities;
using static School.BLL.Infrastructure.AutoMapperConfig;

namespace School.BLL.Services
{
    public class PupilService: EntityService<Pupil, PupilDTO>, IPupil
    {
        IUnitOfWork Database { get; set; }

        public PupilService(IUnitOfWork unitOfWork)
            :base(unitOfWork)
        {
            Database = unitOfWork;
        }


        public override PupilDTO FindById(int? id)
        {
            if (id == null)
                throw new ValidationException("Не установлено id ученика", "");
            var pupil = Database.Repository<Pupil>().Query().Filter(p => p.Id ==
id).Include(i => i.Child).Include(ii => ii.Form).Get().First();
            if (pupil == null)
                throw new ValidationException("Ученик не найден", "");
            return Mapper.Map<Pupil, PupilDTO>(pupil);
        }

        public override void Update(PupilDTO pupil)
        {
            var mapPupil = Mapper.Map<PupilDTO, Pupil>(pupil);
            Database.Repository<Pupil>().Update(mapPupil);
            var p = Mapper.Map<PupilDTO, Child>(pupil);
            Database.Repository<Child>().Update(p);
            Database.Save();
        }

        public override IEnumerable<PupilDTO> GetAll()
        {
            return Mapper.Map<IEnumerable<Pupil>, List<PupilDTO>>
```

```
                (Database.Repository<Pupil>().Query().Include(p => p.Child).Include(i =>
i.Form).Get());
        }

        public IEnumerable<PupilDTO> GetCurrentPupils(int? page, int pageSize, out int
totalPupilCount)
        {
            var pageNumber = page ?? 1;

            var currentYear = DateTime.Today.Year;
            Expression<Func<Pupil, bool>> filterCurrentPupils = s =>
                (s.Form.Year == currentYear && DateTime.Today.Month >= 9 ||
                 DateTime.Today.Month < 9 && s.Form.Year == currentYear - 1)
                && (s.Dismissal == null);

            return Mapper.Map<IEnumerable<Pupil>, List<PupilDTO>>
                (Database.Repository<Pupil>().Query().Include(p => p.Child).Include(i =>
i.Form)
                .SortBy(u=> u.OrderBy(g => g.Child.LastName).ThenBy(g=>
g.Child.FirstName).ThenBy(g=> g.Child.Patronymic))
                .Filter(filterCurrentPupils).GetPage(pageNumber, pageSize, out
totalPupilCount));

        }

        public IEnumerable<PupilDTO> GetPupilsByForm(int formId)
        {
            return Mapper.Map<IEnumerable<Pupil>, List<PupilDTO>>
                (Database.Repository<Pupil>().Query().Include(p => p.Child)
                .Filter(f => f.FormId == formId && f.Dismissal == null).Get());
        }

        public int TransferToForm(int pupilId, int targetFormId)
        {
            var pupil = Database.Repository<Pupil>().GetById(pupilId);
            if(pupil.Dismissal != null)
                throw new ValidationException("Учащийся уже закончил обучение в данном
классе. Отчислить повторно невозможно", pupil.Dismissal.ToString());
            pupil.Dismissal = DateTime.Today;
            Database.Repository<Pupil>().Update(pupil);

            var newPupil = new Pupil
            {
                ChildId = pupil.ChildId,
                Enrolment = DateTime.Today,
                FormId = targetFormId
            };
            Database.Repository<Pupil>().Create(newPupil);

            Database.Save();
            return newPupil.Id;

        }

        public IEnumerable<PupilDTO> GetPupilsBySearchString(string searchString)
        {
            var currentYear = DateTime.Today.Year;
            Expression<Func<Pupil, bool>> filter = s =>
                (s.Form.Year == currentYear && DateTime.Today.Month >= 9 ||
                 DateTime.Today.Month < 9 && s.Form.Year == currentYear - 1)
                && (s.Dismissal == null)
                && (s.Child.FirstName.Contains(searchString) ||
s.Child.LastName.Contains(searchString) || s.Child.Patronymic.Contains(searchString));
```

```
            return Mapper.Map<IEnumerable<Pupil>, List<PupilDTO>>
                (Database.Repository<Pupil>().Query().Include(p => p.Child).Include(i =>
i.Form)
                .Filter(filter).Get());

        }

        public override void Create(PupilDTO pupil)
        {
            Database.Repository<Child>().Create(Mapper.Map<PupilDTO, Child>(pupil));
            Database.Repository<Pupil>().Create(Mapper.Map<PupilDTO, Pupil>(pupil));
            Database.Save();
        }

    }
}
```

```
using System.Collections.Generic;
using School.BLL.DTO;
using School.DAL.Entities;
using School.DAL.Interfaces;
using static School.BLL.Infrastructure.AutoMapperConfig;

namespace School.BLL.Services
{
    public class TermService: EntityService<Term, TermDTO>
    {
        IUnitOfWork Database { get; set; }

        public TermService(IUnitOfWork unitOfWork)
            :base(unitOfWork)
        {
            Database = unitOfWork;
        }

        public IEnumerable<TermDTO> FindByString(string str)
        {
            var terms = Database.Repository<Term>().Where(p => p.Name.Contains(str));
            return Mapper.Map<IEnumerable<TermDTO>>(terms);
        }

        public IEnumerable<TermDTO> FindActiveTerms()
        {
            return Mapper.Map<IEnumerable<TermDTO>>(Database.Repository<Term>().Where(p
=> p.Active));
        }

    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using School.BLL.DTO;
using School.DAL.Entities;
using School.DAL.Interfaces;
using School.DAL.Repositories;

namespace School.BLL.RuleSystem
{
    public class SchoolStatisticInformationService
    {
```

```
        private IUnitOfWork Database { get;}
        private readonly int[] _qualityGrades = { 4, 5 };
        private readonly int[] _progressGrades = { 3, 4, 5 };
        public PeriodDTO Period { get; }
        public IEnumerable<TeacherStatDTO> StatByTeachers { get; set; }
        public IEnumerable<SubjectStatDTO> StatBySubjects { get; set; }
        public IEnumerable<TeacherSubjectStatDTO> StatBySubjectAndTeachers { get; set; }

        public SchoolStatisticInformationService(IUnitOfWork unitOfWork, PeriodDTO
period)
        {
            Database = unitOfWork;
            Period = period;
            SetProgressAndQualityProps(Period.Term.Id, Period.Year);
        }


        public void SetProgressAndQualityProps(int termId, int studyYear)
        {
            StatByTeachers = GetStatByTeachers(termId, studyYear);
            StatBySubjects = GetStatBySubjects(termId, studyYear);
            StatBySubjectAndTeachers = GetStatBySubjectAndTeachers(termId, studyYear);
        }

        public IEnumerable<TeacherStatDTO> GetStatByTeachers(int termId, int studyYear)
        {
            var qualityByTeachers =
TeacherPercent.ConvertList(Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTerm(_qualityGrades, f => f.TeacherId, termId,
studyYear));
            var progressByTeachers =
TeacherPercent.ConvertList(Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTerm(_progressGrades, f => f.TeacherId, termId,
studyYear));
            var teacherValue = qualityByTeachers.Join(progressByTeachers, p => p.Id, t =>
t.Id, (p, t) => new TeacherStatDTO
            {
                TeacherId = p.Id,
                ProgressPercent = t.Percent,
                QualityPercent = p.Percent
            });
            return teacherValue;
        }

        public IEnumerable<SubjectStatDTO> GetStatBySubjects(int termId, int studyYear)
        {
            var qualityBySubjects =
SubjectPercent.ConvertList(Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTerm(_qualityGrades, f => f.SubjectId, termId,
studyYear));
            var progressBySubjects =
SubjectPercent.ConvertList(Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTerm(_progressGrades, f => f.SubjectId, termId,
studyYear));
            var subjectValue = qualityBySubjects.Join(progressBySubjects, p => p.Id, t =>
t.Id, (p, t) => new SubjectStatDTO
            {
                SubjectId = p.Id,
                ProgressPercent = t.Percent,
                QualityPercent = p.Percent
            });

            return subjectValue;
        }
```

```
        public IEnumerable<TeacherSubjectStatDTO> GetStatBySubjectAndTeachers(int termId,
int studyYear)
        {
            var ter = Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTermByTeacherAndSubject(_qualityGrades, termId);
            var qualityBySubjectAndTeachers =
TeacherSubjectPercent.ConvertList(ter).OrderBy(g=>g.TeacherId).ThenBy(h=>h.SubjectId);
            var progressBySubjectAndTeachers =
TeacherSubjectPercent.ConvertList(Database.Repository<GradeForTerm>()
                .GetPercentOfGradesForTermByTeacherAndSubject(_progressGrades, termId));
            var subjectTeacherValue = qualityBySubjectAndTeachers
                .Join(progressBySubjectAndTeachers, p=>new {p.SubjectId, p.TeacherId}, t
=> new {t.SubjectId, t.TeacherId}, (p, t) => new TeacherSubjectStatDTO
            {
                SubjectId = p.SubjectId,
                TeacherId = p.TeacherId,
                ProgressPercent = t.Percent,
                QualityPercent = p.Percent
            });

            return subjectTeacherValue;
        }


        public void Dispose()
        {
            Database.Dispose();
        }


    }
}
```